System-Theoretic Safety Analysis for Teams of Collaborative Controllers

By

Andrew N. Kopeikin

B.S. Aerospace Engineering, University of Illinois Urbana-Champaign, 2006
S.M. Aeronautics and Astronautics, Massachusetts Institute of Technology, 2012

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 2024

Authored by: Andrew N. Kopeikin
Department of Aeronautics and Astronautics
September 15, 2023

Certified by: Nancy G. Leveson
J. C. Hunsaker Professor of Aeronautics and Astronautics
Thesis Supervisor

David A. Mindell
Professor of Aeronautics and Astronautics
Dibner Professor of History of Engineering and Manufacturing

Natasha A. Neogi
Ph.D., Subproject Manager, System-Wide Safety Project, NASA

John P. Thomas
Ph.D., Research Engineer, Department of Aeronautics and Astronautics

Accepted by: Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Disclaimer

This material is based upon work supported by the United States Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the United States Air Force.

System-Theoretic Safety Analysis for Teams of Collaborative Controllers

by

Andrew N. Kopeikin

Submitted to the Department of Aeronautics and Astronautics
On September 15, 2023, in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Human teams collaborate by establishing roles, changing functional authorities, maintaining team cognition, coordinating, and helping one another close control loops. These complex interactions are inspiring novel system concepts to improve human-machine and multi-machine collaboration. However, these new systems challenge existing methods to model, analyze, design, and assure their safety. As such, few have been fielded in safety-critical domains like aerospace.

To address this gap, this work develops a rigorous and systematic approach to analyze safety and enable safety-guided design of systems that exhibit collaborative control. It introduces a system-theoretic framework to describe multi-controller interactions. This includes a taxonomy of seven structural dimensions that influence such interactions and nine dynamics observed in collaborative control that are defined using System-Theoretic Accident Model and Processes (STAMP). An analyzed set of controller interactions in aerospace systems demonstrates the framework and highlights how designers are trying to create more sophisticated systems.

The framework provides the necessary foundation to extend the state-of-the-art in hazard analysis, System Theoretic Process Analysis (STPA), to systematically address collaboration. First, a mechanism is developed to incorporate the nine collaborative control dynamics into STAMP control structure models so that they are explicitly considered in hazard analysis. Second, a process is derived from STPA to identify unsafe combinations of control actions between multiple controllers. The procedure systematically considers potential issues involving gaps, overlaps, transfers, and mismatches in authority that are found in teams. It is executed using an abstraction-based algorithm that manages combinatorial complexity and provides automation support. Third, a method is introduced to identify causal factors from these unsafe control combinations that relate to the collaborative dynamics. The new technique, STPA-Teaming, is applied to a manned-unmanned aircraft teaming case study, and it finds new causal factors not previously found in a past hazard analysis of the same system.

Finally, a structure is derived from Intent Specification to (1) integrate design and assurance processes, (2) support system modeling and analysis at different levels of abstraction, and (3) trace engineering activities using a means-end hierarchy. The framework integrates STPA-Teaming into a broader systems engineering approach. It also leverages the analytical structure of STPA-Teaming to provide novel traceability of its results directly to architectural design decisions. The safety-guided approach is illustrated using the same case study as above.

Thesis Supervisor: Nancy G. Leveson, Ph.D.
Title: J. C. Hunsaker Professor of Aeronautics and Astronautics

Acknowledgments

Completing my doctorate in aerospace engineering has long been an aspiration of mine, but until recently, seemed far out of reach and vanishing given my work and family responsibilities. I am incredibly fortunate to have been given this opportunity. This journey has been both very rewarding as well as challenging, and this dissertation is the result of the support of the following people, without whom I would have been adrift.

First, I cannot convey the depth of my gratitude to Prof Nancy Leveson for welcoming me into her research group and opening my eyes to a whole new view of engineering. Her ideas are revolutionizing how many industries address development challenges and I am privileged to take part in her paradigm change. Prof Leveson spent an enormous amount of time mentoring, developing, and guiding me in my research. Thanks to her, I see the world in control structures, embrace holism, and have solutions to address the most complex systems engineering problems.

My Ph.D. committee included leading experts in their fields and I am especially appreciative of their mentorship. Dr. Natasha Neogi spent countless hours outside her busy role at NASA to meet and carefully review my work. I am grateful for her incredible breadth and depth of ideas, and for the way she challenged me to improve my precision and logic in reasoning. Dr. John Thomas was instrumental in helping me understand the fundamental concepts in my research. His ideas and passion for safety engineering were a key influence on my thought process throughout this journey. Prof David Mindell, whose book on humans and robots inspired my work, provided invaluable perspectives to navigate the often-foggy route of doctoral research.

Other top researchers played key roles during my thesis proposal, defense, and as readers. Dr./Col Bill "$" Young took me under his wing early on and enabled me to apply my research to real-world problems. Prof Elizabeth Baker undertook the massive effort of carefully combing through my dissertation to help me perfect my message. To all my committee members and readers, thank you for sharing your wisdom and time, which molded me into a researcher.

I am grateful for all my interactions with the students in the Engineering Systems Lab. Special thanks to Dr. Lawrence Young for sharing his experiences and to future doctors Michael Schmidt, Justin Poh, and Alex Hillman for our discussions. Also, thank you to those I worked closely with in research efforts: Elias Johnson, Sam Yoo, Dro Gregorian, Adam Munekata, and Brittany Bishop.

I am eternally thankful for the support of the Lincoln Scholar Program. Without it, I could not have earned a Ph.D. while also raising my family. I thank several leaders at MIT Lincoln Lab, including Dr. Marc Viera, Dr. Jenn Watson, Scott Van Broekhoven, and Dr. Caroline Lamb for their professional support throughout my studies. I also appreciate the many conversations I had with Lincoln Lab colleagues that helped frame my research.

Finally, I would like to thank my friends and family who visited and helped us during this journey. Most of all, I thank my two boys and wife for supporting me every step of the way. I love you three more than anything and could not have finished this work without your love. Alex and Aiden, thank you for providing an infinite source of study breaks and for all the joy you have brought me. May this work inspire you to challenge yourselves to aim high and seek new horizons in life, whatever they may be. Ashley (the original Dr. Kopeikin), thank you so much for your support, encouragement, and patience while I spent long days and nights away from you, again, to complete more graduate work. You are forever my inspiration.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivation

A *system* is "a set of components that act together as a whole to achieve some common goal, objective, or end" [1, p. 40]. Aerospace systems components consist of hardware, software, and humans. Human-to-human interactions in systems can be complex. The social process of *teaming* involves dynamics in establishing roles, changing functional authorities, team cognition, coordination, and helping one another close control loops [2]. These mutually influential interactions allow teammates to leverage each other's contributions to joint activity to improve performance in a task or to achieve something they cannot do alone.

The recognized benefits of software control in precision, efficiency, cost, and operational reliability [3], [4] have helped permeate automation in many facets of fielded aerospace systems. However, despite amazing technological progress and expanding applications, the basic interactions between humans and software controllers have remained relatively limited in these safety-critical systems. Most follow the same basic concept. Automated machines control various processes using feedback loops. Humans generally set goal conditions for the control effort, supervise the machines, and in some cases, intervene by changing parameters and modes or by disabling them altogether.

Technological advances, limitations of current systems, and market pressures are all energizing interest in developing designed systems with new types of component interactions inspired by human teams (Figure 1-1). The goal is to extend this social process to enable humans and machines, or multiple machines, to work better together by contributing different strengths and mitigating each other's weaknesses.

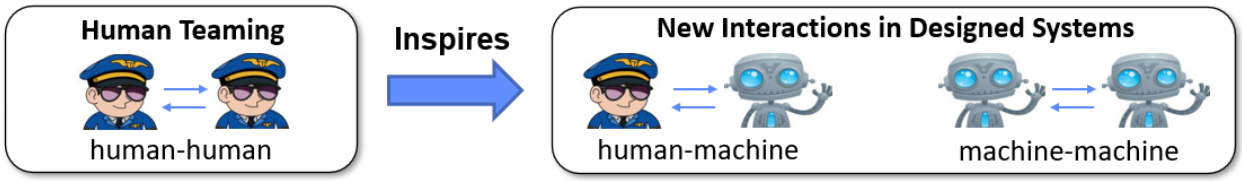


Figure 1-1. Interactions in Human Teams Inspire Designs of New Complex Systems

For example, software controllers, while faster and more precise than humans, are also brittle and cannot handle situations that exceed their programmed bounds or were unforeseen in design [4], [5]. Conversely, by nature, humans are creative problem solvers that can help overcome such situations but are generally poor at serving in passive monitoring roles and cannot effectively intervene if they have been out of the control loop [4]. The outcome of their actions can be improved if the two form a collaborative partnership. Similarly, teams of multiple machines offer opportunities for different systems to leverage potentially disparate capabilities or harness their combined effects to improve their output synergistically.

The past decade has witnessed numerous proposed concepts to implement these more complex interactions in aerospace systems. For example, the potentially $500B market of Urban Air Mobility (UAM)[1] [6], [7] is confronted with an anticipated pilot shortage [8], and its demands are expected to exceed human pilot and air traffic controller workload capacities [9]. As such, some in the aviation industry envision partnering humans with future automated controllers that would be certified to safely take on some of the operating responsibilities [10]. The industry is also exploring how teams of ground-based human operators can share remote control of multiple UAM aircraft, possibly without any vehicle operator on board [6], [8].

Other aerospace organizations are exploring similar concepts. The airline industry is studying whether flight crews can be reduced to a single pilot using support from onboard automated systems [11], [12] or from ground-based human-machine systems [5], [13], [14]. Civil aviation authorities are researching how distributed decision-making and dynamic switching between human and automated controllers can help integrate UAM and Unmanned Aircraft Systems (UAS) traffic into the National Airspace System [8], [9], [15]–[17]. Similarly, the space industry is seeking to implement distributed control of new multi-satellite constellations [18], and it is exploring how human-robot partnerships improve deep space and planetary exploration [19].

Defense sector roadmaps show similar ambitions in improving how autonomous systems partner with humans and other machines [20], [21]. For example, new defense system goals include collaboratively pairing pilots with automation that acts as an additional crewmember and dynamically offloads some operating tasks [22], [23]. Numerous early prototypes of distributed control multi-UAS and multi-munition systems have demonstrated they can achieve collective group effects [24]–[27]. Several conceptual designs include teaming human-piloted aircraft with UAS to jointly execute complex missions [28]–[31]. Finally, some argue the military should increase its use of smaller collaborative heterogeneous unmanned systems to deliver more overwhelming effects [32].

Despite the high interest in engineering aerospace systems with these more complex interactions, few (except for the simplest designs) have actually been fielded. Design options are constrained by a gap in the existing systems engineering processes, which cannot effectively model, analyze, design, and assure the safety of such systems. The goal of this dissertation is to solve part of this gap by developing a novel framework that supports the analysis and design of systems with degrees of freedom in component interactions that go beyond those fielded today in aerospace.

[1] Urban Air Mobility (UAM) is part of the larger Advanced Air Mobility (AAM) concept, which envisions large numbers of increasingly autonomous aircraft operating densely in airspaces to transport people and goods [6]. The terms UAM and AAM are used interchangeably in this work.

1.2 Challenges

Safety in aviation systems is non-negotiable and is arguably the most critical constraint to satisfy. If an aviation system is viewed as unsafe, it will not be accepted by society and will not be adopted, regardless of how well the system performs in other aspects.

Historical trends in aviation indicate that the introduction of new generations of automated technologies is typically immediately followed by an initial increase in accident rates [33]. To improve safety, potential unsafe causal factors must be systematically identified, understood, and eliminated or mitigated over the life cycle of the system. New systems that involve more complex human-machine and multi-machine interactions will be no different.

This section first describes some of the challenges observed in relevant contemporary systems that have led to accidents. These systems include the simpler interactions found in modern supervisory control of automation, as well more complex interactions in teams of human operators. Next, it explores the numerous open questions expressed in the literature regarding how to design more collaborative relationships into systems.

1.2.1 Relevant Challenges in Contemporary Systems

Aviation's history is fraught with unsafe interactions involving human supervisory control of automated systems. These are important to consider as they form a subset of the expanded types of relationships being considered in future designed systems.

In 2009, Air France flight 447 (AF 447), with its fully functional Airbus A330 and highly trained flight crew, lost control in cruise, crashed, and killed all 228 people onboard. A temporary blockage of the airspeed sensors led the aircraft automation to change flight-control modes and displays and created confusion in the cockpit on how to interact with the aircraft [34], [35]. In 2013, a Boeing 777 crashed short of its landing runway when the flight crew issued manual flight control inputs on a diverged mental model of the auto-throttle operating mode [36]. More recently, several Boeing 737-MAX flight crews had to "fight" unsafe automated flight control inputs to regain aircraft control. In these cases, design defects and certification oversights that occurred in market-driven haste to field the new aircraft contributed to two crashes, 346 lives lost, and a costly worldwide grounding of the new aircraft fleet [37].

Contemporary aviation systems also include human teams, which exhibit the complex dynamics of interest, but in doing so, have also contributed to accidents. In the same AF 447 example, the confusion that arose in diagnosing the flight control system also led to a catastrophic breakdown in teamwork within the crew of three. Inadequate coordination and rapid changes in control authority resulted in two pilots issuing simultaneous opposite control inputs, which canceled each other out [35]. The system was not designed to prevent getting into such a hazardous state from this contributing factor.

Military aviation has also suffered from unsafe human teaming. In 1994, two US Air Force F-15s accidentally shot down two US Army Blackhawk Helicopters, killing all 26 personnel onboard. Some of the causal factors of this friendly-fire accident relate to unsafe teamwork between two airborne controllers on the AWACS (Airborne Warning And Control System) that

managed the Iraqi No-Fly Zone [38]. Overlaps, gaps, and evolutionary changes in control authority of the air traffic resulted in an unsafe collective output from the AWACS controller team.

1.2.2 Open Challenges to Address

The community is becoming increasingly aware of some of the difficulties associated with expanding the nature of component interactions in aerospace systems. Several recent studies describe key challenges and research gaps [8], [23], [39]–[44], and highlight three recurring topics: (1) it is challenging to engineer human-team inspired interactions, (2) there is a need for new associated design methods, and (3) existing safety assurance processes are unable to address these more complex interactions. The state-of-the-art in modeling, design, and analysis for each of these topics is further reviewed in Chapter 2.

All eight studies describe the difficulty of engineering the attributes that exist in human teams into systems that facilitate safe teaming between humans and automation. System components (humans and machines) may need to handle context-dependent dynamics such as establishing and transitioning roles, responsibilities, and functional authorities. Collaborators may have to establish and maintain shared situational awareness of a joint control problem. Controllers, both human and automated, may help close each other's control loops and perform mutual monitoring. These dynamics are further complicated by inherent differences in how controllers interpret information and time, especially when humans partner with machines.

Other attributes, which current systems already struggle with, may be further challenged by more complex interactions. The level to which humans trust machines and understand how they work heavily influences their interactions [45]. The age-old open question of how to effectively keep humans in the control loop when working with automation also remains. As more functions become reliably automated, it is less likely human operators working with them will be able to detect off-nominal events or recover control [4], [44]. More dynamic and interdependent relationships between humans and automation can further strain these issues.

The next recurring topic highlights how current design techniques fall short in engineering more complex systems. There is a clear need to improve how interdependence is modeled between system components. Systematic and rigorous processes are also lacking to guide architectural decisions, such as functional allocation [39]. The community increasingly recognizes that the traditional reductionist approaches commonly used are inadequate to design complex and safety-critical systems. For instance, the typical method of *divide and conquer* does not account for interactions between entities [38], [46], and the more recently popular practice of deriving design decisions based on a *Level-of-Automation* number is too simplistic to handle real-world complexities [47]. More sophisticated methods (see Chapter 2) have their own challenges and limitations and, therefore, have not been operationally applied in aerospace.

The last common theme centers on our inability to assure novel aerospace systems. Assuring safety requires a holistic systems engineering approach that spans requirements generation, design, verification and validation (V&V), certification, operations, and evolution. The studies point to challenges associated with developing requirements and metrics for collaborative systems that are end-to-end traceable and analyzable. They all describe shortfalls in traditional V&V methods, which are challenged against complex software-intensive systems, do not address

the influence of dynamic partnerships with automation on human performance [48], and are unable to validate if a system will degrade gracefully.

In addition, many of the new technologies proposed are not addressed by prescribed certification standards. For example, many new systems want to incorporate recent advances in Machine-Learning, which employ non-deterministic and adaptive algorithms that have never been successfully certified in safety-critical applications. Furthermore, the aviation regulatory framework is predicated on a human pilot having final authority and responsibility for the operation of an aircraft [49]. This premise creates tension with concepts that shift some of these responsibilities over to autonomous functions. Finally, the significant costs of assurance activities can be prohibitive to new, and often small, manufacturers of future UAM aircraft or UAS that have limited resources.

1.3 Research Overview

The goal of this dissertation is to advance the state-of-the-art in system safety engineering to address some of the challenges listed in Section 1.2.2. The following describes the overall research objective, gap, contributions, hypotheses, and scope.

1.3.1 Research Objective

The objective of this research is to develop a rigorous and systematic framework that enables safety analysis and safety-guided design of systems that exhibit collaborative control interactions.

1.3.2 Gap

The aerospace industry is clearly interested in engineering systems that enable "teaming" between humans and machines and among multiple machines. While the term "teaming" implies something new, it is vague and used to describe many different types of concepts, much like other buzzwords. What it truly implies is that the community is pursuing system designs that push the boundaries in the complexity of component interactions beyond what is currently fielded.

Despite the interest, there is a distinct gap in the ability of systems engineers to describe the different types of component interactions. Current modeling techniques are inadequate to account for some of the dynamics observed in human teams that are inspiring new designs. Finally, there are no rigorous and systematic processes to define and analyze system architectures with these more complex interactions, nor to assure their safety in a cost-effective way [39].

Analysis techniques based on System-Theoretic Accident Model and Processes (STAMP) have successfully guided the design of systems and their emergent properties, such as safety and security, starting early in the engineering lifecycle. STAMP is an accident causality model grounded in Systems Theory [38]. It enables systematic analysis of non-linear causal relations between hardware, software, and human controllers that interact in systems. Its use of

abstraction manages complexity, and its hierarchical framework helps reason about socio-technical factors holistically and top-down. System-Theoretic Process Analysis (STPA) is a hazard analysis method built onto STAMP that has become popular in many industries because of its ability to identify potential causal factors early in design [50].

STAMP and STPA have attributes well-suited to address some of the challenges associated with fielding the novel aerospace systems introduced earlier. However, they lack a framework to systematically consider the collaborative dynamics sought in these systems. This dissertation aims to address this gap to improve the analytical performance of these sophisticated techniques.

1.3.3 Research Contributions

Contribution 1: *The system-theoretic definition of interactions observed in collaborative control.*

A widely cited definition for *team* is: "A team consists of two or more entities who interact dynamically, interdependently, and adaptively toward a common and valued goal, with unique roles and functions to perform" [51, p. 3]. It is nearly identical to that of a *system* (see Section 1.1) and, therefore, is not very useful by itself in distinguishing teams from other systems.

What makes teams different and so challenging to engineer is that their component interactions are more complex than those in previously fielded designed systems in safety-critical applications. Despite the strengths of system-theoretic methods to analyze complex systems, many of the types of interactions that may be designed into systems have not been defined using Systems Theory. This dissertation provides a framework to define interactions observed in collaborative control so that they can be more completely analyzed using STAMP and the analysis tools built on it.

The system-theoretic framework consists of (1) a taxonomy of the structure of interactions between multiple controllers and (2) a set of dynamics observed in collaborative control. It creates the necessary foundation to extend system-theoretic hazard analysis methods needed to systematically identify causal factors associated with these interactions.

Contribution 2: *Extensions to STAMP and STPA that enable systematic analysis of safety in systems that exhibit collaborative control interactions.*

STAMP and its analysis tools need extended guidance to systematically handle the more complex, team-inspired component relationships sought in novel systems. Current STAMP models tend to represent systems with rigidly assigned control authorities. The procedure in STPA lends the focus of analysis to one controller and one process at a time. These methods do not emphasize complex dynamics in collaboration, such as shared process models, joint control, and shifting roles and responsibilities.

This dissertation introduces several extensions to STAMP and STPA to systematically identify causal factors associated with collaborative control. First, a generic collaborative control structure provides a mechanism to incorporate collaborative interactions in STAMP models. Second, a process is established to identify unsafe combinations of control actions between multiple controllers. The procedure systematically considers potential issues involving gaps, overlaps, transfers, and mismatches in authority that are found in teams. Finally, a method is designed to identify causal scenarios from these unsafe control combinations that are guided by the system-

theoretic definition of collaborative interactions (Contribution 1). These extensions are collectively referred to as STPA-Teaming.

Contribution 3: *A framework to integrate safety-guided architecture design with assurance through enhanced traceability.*

Safety assurance processes are typically conducted separately from design and not until later stages of development. By the time they are applied, it is often impractical to (1) modify the system if safety issues are found, and (2) perform effective V&V to ensure hazards are properly eliminated or mitigated. One of the strengths of STPA is its ability to analyze systems throughout their engineering lifecycles, including in early conceptual design stages. This allows safety requirements to be identified early when they are most useful and can establish traceability that supports a more effective assurance by construction program [4].

This dissertation provides a framework derived from Intent Specification [52] to help integrate design and assurance. The framework enables navigation and traceability between three axes. First, the results of the hazard analysis are traced to derived safety constraints and then to the selected V&V strategy. Second, the system is modeled at different levels of abstraction, which represent a team as a whole at a higher level, but also capture collaborative interactions within the team at a lower level. And third, the system is described using different views in a means-end hierarchy, starting from concept of operation, down to low-level component implementation.

Given the focus of this research, the dissertation emphasizes how Contributions 1 and 2 integrate into the framework, on all three axes, to support conceptual design decisions regarding collaborative control systems. However, a similar construct may be generalizable to other types of systems beyond those that exhibit complex team-inspired interactions.

1.3.4 Hypotheses and Evaluation

The research explores the following three hypotheses. Their evaluation supports an argument toward validation of Contributions 1 and 2 to the state-of-the-art provided by this dissertation. Contribution 3 includes a demonstration of the vision for how this work fits within a broader systems engineering context, but no formal evaluation is provided.

Hypothesis 1

The system-theoretic collaborative interactions framework provides a mechanism to categorize and describe component interactions that are, or planned to be, designed into aerospace systems.

The evaluation of this hypothesis is conducted in Chapter 3 over a set of fielded and unfielded aerospace systems reviewed in the literature. It provides a demonstration of how to categorize interactions within these systems using the framework. Analysis of the categorized interactions supports the conclusion that the component interactions are being sought.

Hypothesis 2

The system-theoretic collaborative interactions framework describes component interactions that are not specifically addressed by existing hazard analysis techniques, including STPA.

This hypothesis is evaluated in Chapter 4 by qualitatively showing that one or more of the component interactions identified in the framework, which are sought out in the design of aerospace systems, are not systematically addressed in the existing hazard analysis techniques.

Hypothesis 3

The STAMP and STPA extensions identify causal factors associated with collaborative control interactions, which are not systematically found using the existing STPA guidance.

This hypothesis is evaluated in Chapter 5 using a case study of a real-world system concept involving Manned-Unmanned Teaming (MUM-T). The system involves a human-piloted military aircraft that collaborates with multiple UAS to execute mission tasks. The evaluation includes two parts.

First, the extended hazard analysis technique, STPA-Teaming, is performed on the same MUM-T concept that was previously and independently analyzed using STPA [53]. The evaluation demonstrates how new scenarios related to collaborative control are uncovered using the analytical extensions developed in this work. However, the system in this case study does not exhibit all of the collaborative interactions defined in this dissertation.

The second part of the evaluation fills this gap. The original case study is expanded by hypothesizing new collaborative control interactions that could be incorporated into MUM-T. Hazard analysis of the new concept completes the demonstration that STPA-Teaming finds causal scenarios associated with all of the collaborative control dynamics defined.

1.3.5 Scope

The research is scoped in three ways to ensure timely completion with the appropriate depth and level of contribution of a Ph.D. dissertation. First, the analysis of collaborative interactions is restricted to safety. Here, safety is defined as the *absence of unplanned and unacceptable losses* [38]. These losses may include loss of life, injury, or damage to property, as often described in accident reports. However, they can also include broader items, such as loss of mission, loss of reputation, etc., which may also be unacceptable to stakeholders for some systems.

There can be numerous and often competing objectives in design. As previously described, safety is arguably the most important constraint to satisfy in aerospace applications. However, it does not necessarily reflect other figures of merit that influence design optimization. In addition, while STAMP and STPA can address other emerging system properties, like security, these are not the focus of this work.

Second, the analytical extensions target a subset of all possible system interactions. They specifically concentrate on those observed in collaborative control as defined in this work. Many other types of component interactions are well-studied in the STAMP literature. For example, feedback control loops for supervisory control or automated decision aid systems are well

understood in STAMP and do not need to be redefined here. The goal is to enable analysis of system designs that go beyond these more traditional relationships. However, it is also possible that other types of interactions beyond those identified in this dissertation may be proposed. As such, this work aims to address the most important and distinctive aspects needed to extend STAMP and STPA for collaborative control.

Third, this research focuses on designed systems in the aerospace domain. It is expected that many of the techniques developed will also apply to other fields. However, the set of collaborative interactions studied is bounded to those anticipated in aerospace.

1.4 Organization of Dissertation

The remainder of this dissertation is organized as follows.

Chapter 2 reviews and evaluates the literature relevant to the research objective. It describes the state-of-the-art in modeling, designing, and analyzing systems inspired by teams. The literature reviewed includes theoretical foundations of teaming interactions, methods used to design their architectures, and assurance processes to get them fielded. The chapter also provides the necessary background in Systems Theory, STAMP, and STPA to justify their use and the need for analytical extension to address the research gap.

Chapter 3 introduces the system-theoretic framework to analyze collaborative control interactions. It defines a taxonomy of the structure of interactions between multiple controllers and a set of dynamics observed in collaborative control. The chapter applies the framework to a sample of interactions found in fielded and unfielded aerospace systems to demonstrate its use and support its relevance.

Chapter 4 leverages the theoretical foundation from Chapter 3 to develop extensions to STAMP and STPA that enable systematic analysis of causal factors associated with collaborative control. The new techniques include guidance to model collaborative control structures, a mechanism to identify when control actions by multiple controllers are unsafe together, and a method to develop causal scenarios framed by the collaborative dynamics.

Chapter 5 applies the extended hazard analysis technique to a case study of a real-world aerospace concept. First, the new method is applied to the same system previously analyzed using STPA to demonstrate how it finds new causal scenarios. Next, the concept in the case study is expanded to demonstrate how the new technique addresses all of the types of collaborative control dynamics defined in this work.

Chapter 6 describes a framework derived from Intent Specification that integrates safety-guided design with assurance processes using the results of the extended hazard analysis techniques from Chapter 4. A demonstration illustrates how the framework can support safety-guided design by rigorously deriving and tracing safety considerations to potential design decisions.

Chapter 7 concludes by summarizing the dissertation, acknowledging its limitations, and recommending future work to expand this research.

Chapter 2: Literature Review

This section reviews the state-of-the-art in the literature related to the research objective. The following concepts are necessary to have engineering control over modeling, analyzing, designing, and assuring the safety of systems with complex interactions inspired by human teaming. Four perspectives are explored.

Section 2.1 studies the theoretical foundations available to model and analyze interactions in teams. Section 2.2 examines the methods employed to design architectures that seek these interactions. Section 2.3 explores the processes available to assure the safety of such systems. Finally, Section 2.4 reviews the relevance of system-theoretic approaches to engineer these systems.

2.1 Theoretical Foundations of Team Interactions

Theoretical research on teaming originated as a human-centered study. In 1955, Marschak published the first known "Theory of Teams" to describe interactions between teammates [54]. The field has since produced a significant body of research devoted to better understanding how multi-person teams function.

More recently, a new community has extended the human teaming knowledge base to explore how humans can collaborate with automated software-controlled systems. In parallel, a separate research area has focused on distributed and collaborative control of multiple unmanned systems. Many of their component interactions are relatable to those found in human teams.

It is important to understand the similarities in the interactions between these different types of entities to develop a unified high-level framework for analysis and design, regardless of the system composition. Similarly, differences must also be understood to enable refinement once lower-level details are needed. To this end, this section surveys theoretical principles associated with human, human-machine, and multi-machine teams. These lay the foundation for models needed to design architectures and assure safety for systems with such complex component interactions.

2.1.1 Human Teams

Sports teams, music bands, professional partnerships, and small military units are a few examples of the multi-person teams ubiquitous in society. Teams in aviation include flight crews, flight formations, pilots interacting with air traffic controllers (ATC), and multiple ATC facilities collaborating to manage traffic.

Several accidents in the 1970s stressed the importance for airline flight crews to function as a team instead of as a set of individuals with technical expertise in operating aircraft [23]. This fueled a surge in teaming research [55] and launched the concept of Crew Resource Management

to train pilots on the effective teamwork necessary for flight operations [56]. The resulting body of knowledge summarized below focuses on attributes in team models, theories of team cognition, and integration of teams within larger systems.

The foundational literature on teaming is mostly grounded in Human Factors and Organizational Psychology. It describes teaming attributes in many different ways. For example, MIT's Humans Systems Engineering course associates teams with *association, interdependence, communication, interaction*, *influence*, and *structure* [57]. In another example, Paris et al. explain how teammates must handle *multiple sources of information, task interdependence, coordination and communication between members, common valued goals, specialized roles and responsibilities, task-relevant knowledge,* and *adaptive strategies* [58]. The same authors also acknowledge that the literature provides a vast variety of different taxonomies and models, which have varied in emphasis over the decades.

This wide disparity of models is also echoed in a well-cited paper by Salas et al. [59] who surveyed 138 related studies on teams. To provide focus, they distill the results of their survey into a set of "five core components" needed for effective teamwork. First, *Team Leadership* is needed to direct and coordinate the activities of other team members, assess overall performance, reassign tasks, and synchronize contributions. Second, *Mutual Performance Monitoring* helps develop common understanding and identify mistakes or lapses in teammates. Third, *Backup Behaviors* enable responsibilities and workloads to be shifted. Fourth, *Adaptability* allows the team to identify a need to change strategy and turn to a backup behavior. And Fifth, *Team Orientation* is an attitudinal attribute describing the willingness of a member to work in a team and accept feedback from teammates [59].

Salas also highlights that the five core components must be supported by three "coordinating mechanisms": S*hared-Mental Models*, *Mutual Trust*, and *Closed-Loop Communications* [59]. This model, visually represented in Figure 2-1, represents the state-of-the-art in defining the elements of effective human teamwork from an organizational psychology perspective.

The more specific study of team cognition has also generated a lot of interest. There are two complementary theories to describe how teams establish situational awareness (SA) [57]. The first, developed by Endsley [60], focuses on *shared* elements of SA. In this model, teammates form overlaps in SA to coordinate in joint activity. These overlaps define the information that must be shared, or held in common, between different members. The second model, proposed by Stanton et al. [61], considers SA as *distributed*. Here, instead of focusing on common knowledge, the emphasis shifts to knowledge the system possesses as a whole and whether teammates have SA of who on the team knows what and when. This process drives coordination requirements.

These two theories are now viewed as complementary, where team cognition includes both shared-mental models (models of activity and environment held in common) and *transactive memory* [57]. The latter term refers to benefits that can be obtained on a team when there are useful divergences in knowledge between teammates, and when teammates have SA of who knows what [62].

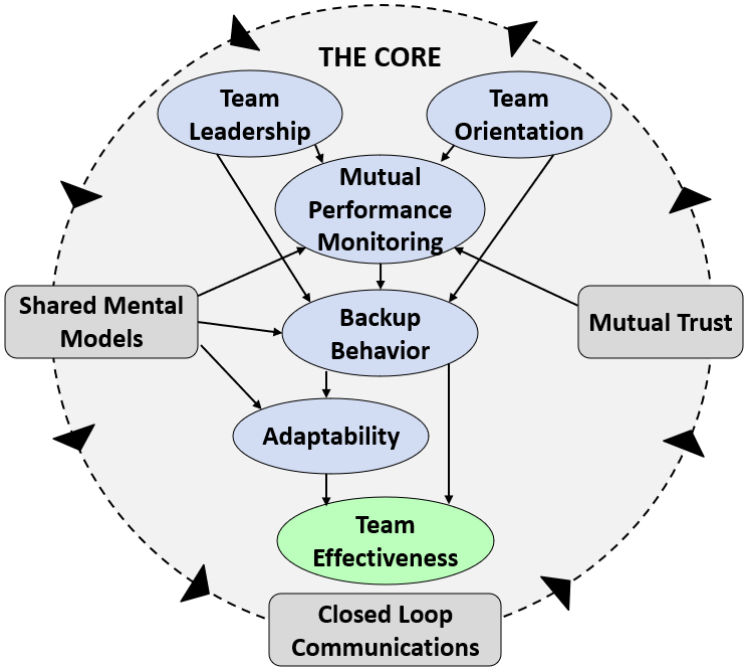


Figure 2-1: Model of effective human teamwork (adapted from [59])

Another common theme in the literature, which is relevant to this work, is that "teams are more than collections of individuals, and teamwork is more than the aggregate of their individual behaviors" [58, p. 1]. This depicts teaming as an emergent system property, which can therefore be analyzed from a system-theoretic lens. While this concept is existent in the literature, the models that claim to use such an approach only do so superficially. They treat the team as an open system with inputs to the team, team processes, and outputs of the team [58], [61] but do not address the core theoretical principles of Systems Theory (see Section 2.4). Most importantly, these models provide little guidance on how to analyze the team when integrated into a larger system context.

This last topic is touched on in Ilgen's research summary on the behavior of teams embedded within larger organizations [55]. In it, he notes that the consideration for the larger system increases the analytical emphasis on inputs and outputs rather than just focusing on team processes. More notably, Kozlowski & Salas [63] employ Organizational Systems Theory to model the transfer of team training between multiple hierarchal levels: organization, team, and individuals. The authors note that the use of hierarchy and emergence helps uncover training issues that would otherwise go neglected. However, their model is specific to their training topic, and no path is provided to generalize the approach to a broader analysis of team dynamics.

In summary, this body of research provides useful considerations to understand how teams function, but some gaps require further investigation. First, the majority of the models surveyed by Paris [58] and Salas [59] are aimed at improving various aspects of team performance, but none of them focus explicitly on ensuring safety. Second, these studies rarely provide tangible guidance to systematically analyze or design a team as a system. Many of them provide a model but no instructions on how to use it. Third, the available direction is even sparser on how to analyze or design a team that interacts within a larger socio-technical context, and no general framework to guide such activities is available. Finally, the models in this section are limited to

interactions between humans only. The implications of introducing machines as teammates are explored in the next two subsections.

2.1.2 Human – Machine Teams (HMT)

The Human-Machine Teaming (HMT)[2] research community has derived much of its theory from the study of human teams described above. Its literature is summarized below with the intent to (1) highlight different types of Human-Automation Interactions (HAI), (2) identify attributes of HMTs important in modeling, and (3) distinguish the fundamental differences between HMTs and human-only teams.

The concept of HMT is interpreted in many different ways. To some, it loosely represents any human interaction with a machine, and therefore a person using a simple gas meter can be classified as an HMT, as is done by Stanton [61]. However, most have higher expectations and sometimes label machines that do not rise to the level of teammates as *tools* [64], [65].

Novel HMT concepts generally seek to shift the paradigm of HAI from the traditional human supervisory control model over to more collaborative partnerships. These partnerships are still generally expected to be human-directed [39]. However, concepts are also theoretically possible where machines may supervise humans [60], [66]. Furthermore, Mixed-Initiative Interaction systems even consider how this hierarchy may dynamically change during operation given different types of collective work [67].

Mosier et al. provide a useful orientation to different types of HAIs found in aviation autonomous technologies [23]. In their work, Information Automation technologies such as Enhanced Vision Systems (EVS), Traffic Collision Avoidance Systems (TCAS), navigation systems, and decision support systems are considered *tools* with limited capability to function as a teammate. They then classify supervisory control systems as more proactive, human-initiated and supervised, and as including a sense of shared mental models and collaboration [23]. Automated copilots like the one described by Dropkin et al. [22] are exemplified as such systems.

They then describe novel HMT concepts as forming even tighter interdependence between humans and automation. Distinguishing features include mutual monitoring, providing feedback to each other, and adapting together dynamically [23]. Early prototypes of these systems exist [2], [68], but none have actually been fielded in aerospace [23], [39]. While these categorizations are helpful to differentiate some HMTs from more basic HAIs, the delineation is still left too ambiguous to determine fundamental architectural differences.

The literature offers many different lists of HMT effectiveness characteristics to include in modeling, much like what was found in human teaming. Some common characteristics distilled from the following studies [5], [13], [43], [60], [69] include *bi-directional information flow*, *shared and team cognition*, *coordination* and *collaboration*, *shared authority*, *shared goals*, *directable* (*human directable* [5], [13]), and *automation transparency*. These are considered in a variety of modeling

[2] *Human-Automation Teaming (HAT) is also commonly used and is interchangeable with Human-Machine Teaming (HMT) in this work.*

techniques, including look-up tables, finite-state machines, network theory, and computational operator function models [70].

Unsurprisingly, many of the proposed performance attributes are drawn from the human teaming theoretical foundations described previously. For example, Battiste et al. describe how they leveraged key principles of Crew Resource Management (CRM), the 40+ year-old benchmark for effective aircrew teamwork, to design an automated decision aid for a ground control station operator [5]. Similarly, Mosier et al. derive key HMT characteristics from the "five core components" and "coordinating mechanisms" previously discussed by Salas [59] (see Figure 2-1), and translate their meaning to the context of machine teammates [23].

However, it is insufficient to rely just on human factors and organizational psychology research in human teaming to engineer a safe HMT system. Automated software-based entities are not sentient beings, and concepts like knowledge, awareness, trust, or even intelligence do not apply to them. As Klein et al. state: "the inherent asymmetry in competencies between people and machines will always create difficulties for designing Human-Automation teams" [69, p. 93].

In fact, some have suggested that automation should never be labeled as a teammate since it lacks affective and cognitive processes comparable to those of humans [71]. For example, machines do not have a sense of responsibility, motivation, loyalty, or values to guide critical problem-solving [39]. Unlike humans who employ creativity when they face the unknown, machines are unable to perform beyond their programmed bounds [4]. They do not have good capabilities to anticipate the needs of the team, particularly those of humans [39], [69]. They handle information processing and timing differently than humans do [42]. Machines are challenged by both syntactic and semantic nuances that humans can handle in natural languages [65]. Finally, machines lack expected team etiquette and unduly interrupt humans [5], [39].

Many of these issues are the subject of dedicated research topics and an integrated approach will be required to handle diverse technical disciplines. Clearly, HMT models must be able to account for differences from human-only teams.

Trust-based effects are also different in HMTs. Trust is unidirectional as only humans can trust automation, and not vice-versa. It is a key factor in determining how a human interacts with automation or intervenes. Under-trust in automation leads to disuse when it would actually benefit team performance. Over-trust can lead to misuse and abuse where team performance can degrade or a hazard can arise [45]. A human that does not trust the automation must allocate resources to checking it as opposed to collaborating with it to accomplish team goals [59].

These effects have fueled calls to make automation transparent and explainable. When automation lacks transparency, aviation operators face difficulty in answering "What is the automation doing? Why is it doing it? What will it do next?" without being overwhelmed with large quantities of information [3].

This is particularly challenging when using non-deterministic Machine-Learning (ML) techniques that have permeated many automation applications. For example, a recent study showed that humans preferred to collaborate with a more predictable rule-based software system, rather than with a "state-of-the-art" ML system with higher stand-alone performance [72]. Similarly, in another study where humans were trained to understand when an ML decision aid was prone to making good or bad decisions, researchers found mixed results in the ability of humans to properly calibrate their trust in such systems [73].

Different models of trust exist in the literature. For example, Muir models trust as an aggregation of different factors that allows it to progress through stages of *predictability*, *dependability*, and *faith* [74], [75]. Cofta proposes another model in which trust and control are two mechanisms for one to gain confidence in another entity: if they do not trust it, they must be able to control it to have confidence in it [76]. Lee & See handle trust as a compilation of three information bases regarding *what*, *how*, and *why* an automated system behaves a certain way [77].

NASA connects Lee & See's model [77] to the human operator mental model in an analysis of HMT effectiveness [78]. They then propose to assess operator mental models using various measurements to adapt the system to better calibrate human trust in it, either through design iterations and experimentations or during live operations. Unfortunately, those ideas are too conceptual and unvalidated to be of any use at this stage.

Despite all the interest, the literature also recognizes that trust in HMTs still needs a better definition and more precise analysis mechanisms to be useful in guiding design decisions [70]. It is noted that the relationship of trust to Control Theory in Cofta [76], and mental models in Lee & See [78], provides a pathway to integrate trust into a system-theoretic framework, such as the one introduced in Section 2.4.

Another key challenge to address in HMTs is effectively maintaining the human in the control loop. Humans are needed in complex system operations because of their creative and adaptable behavior [4]. However, their ability to serve as effective monitors and backup controllers to automation is inherently limited. Human monitoring vigilance cannot be maintained long, especially in the presence of perceived reliability, and it can be difficult to recognize that the automation is faulting if feedback is only coming from the automation itself [4].

Humans also face difficulties in intervening for several reasons [4]. They need time to rebuild an accurate mental model of the system state to recover it. Humans are also often tasked with recovering from difficult situations. Finally, their control skills may be diminished due to lack of exercise because of the autonomy. This is also known as the "automation conundrum": as more functions become reliably automated, the less likely human operators will be able to detect and recover control [44]. This phenomenon is described as a "fundamental barrier" in human supervisory control of automation, and it is a motivator to expand the types of interactions to be more collaborative in nature [4], [39], [44].

The HMT literature has similar gaps to those found in human teaming. There are many different models with many different modeled attributes, and most are focused on effectiveness rather than safety. They generally provide very little direction on how to use the models for analysis or design. The models do not clearly determine if there are fundamental differences between different types of HAIs, and which interactions specifically go beyond those traditionally fielded in aerospace systems. Finally, the literature does not rigorously address how to analyze HMT interactions with the environment, higher-level socio-technical systems, and lower-level components.

2.1.3 Machine Teams

The study of collaborative behaviors achievable by multiple machines originated in controls [79] and computer science [80]. Despite having different theoretical origins than research in human

teams or HMTs, these systems exhibit complex interactions and dynamics that are relatable to those involved in teamwork. The following is a brief review of multi-machine architectures, how some of their system attributes relate to the previously surveyed teaming literature, and the general gap observed in this basis of the literature.

The concept of executing a complex mission using a collection of robots has generated an entire field of research, as illustrated in one recent survey on multi-Unmanned Aircraft Systems (multi-UAS) studies [81]. Coordinated control of these systems can either be centralized, where a single controller determines all the actions for all other systems, distributed, or some hybrid of the two [82]. In distributed control, each individual system makes its own decisions about its actions based on its goals, a reward function, and information shared with its peers [80]. Distributed architectures are most beneficial in highly complex and dynamic environments since they reduce computational latencies and can handle communication losses better than centralized solutions [83].

Many key attributes that enable distributed control of unmanned systems are relatable to those previously described in human teams and HMTs. Like other teams, distributed systems must be directable and share a variable common set of goals [80]. The very concept of distributed control inherently relates to that of shared authority in teaming because multiple controllers can influence a common process.

Distributed systems also rely on coordination and collaboration to achieve synergistic effects [80]. This requires bi-directional information exchange between systems [83]–[86], whether through active messaging, or passive observation of one another [79]. They employ algorithmic processes to reach information consistency, or consensus [83], [84], [87], which is strongly related to the concept of team cognitive alignment. Finally, they must do all this while being subjected to time delays and asynchronization [87], [88], as observed in other types of teams.

While there is a very technically deep literature base for all the distributed control topics described above, much of it treats these systems as fully autonomous and does not consider the human interaction aspect. However, complete automation is a myth, and the safety of these technologies must account for the human roles and ensure robot actions reflect their intentions [34]. The challenges of human-in-the-loop control and trust play a greater role in these systems than is often acknowledged, regardless of whether the human interacts as a "teammate" or as a "supervisor".

In addition, most of these studies are focused on performance optimization rather than safety. The few examples that do emphasize safety [89]–[91] typically address it using a narrow definition and lack the rigor required for safety-critical systems. Finally, the algorithmic nature of these works often provides no path to analyze the system when integrated with its higher-level socio-technical context. These shortfalls are likely part of the reason why these systems have not yet been operationally fielded despite the high interest in them.

2.2 Architecture Design for Collaborative Systems

A system architecture has a strong influence on how a system as a whole will behave in its intended functions [92]. For this reason, it is critical that the design of novel aerospace teaming

system architectures be safety-guided. This section reviews the fundamental principles of system architectures, techniques used to analyze the collaborative functions they enable, and the current state-of-the-art methods to guide their design.

2.2.1 Introduction to System Architectures

The following discussion defines what a system architecture is, describes its role within the development lifecycle, and highlights challenges that arise using common architecture development techniques. This provides the foundation needed to (1) scope the meaning of architecture with regard to collaborative control systems and (2) review analysis and design techniques employed specifically for them.

In basic terms, a system architecture is a model of the system entities and the relationships between them [92]. In the traditional *V-model* of systems engineering, architecting is a design process that occurs on the left side of the "V" (Figure 2-2). Its inputs are high-level system requirements generated from early conceptual processes such as the stakeholder analysis and Concept of Operations. Its outputs then feed the more detailed design requirements of the system and its components.

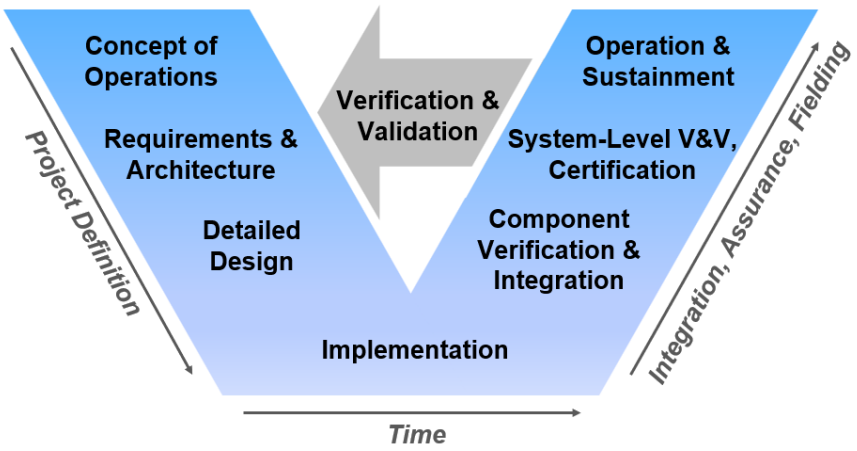


Figure 2-2: Systems Engineering "V-Model" (adapted from [93])

Ideally, the system architecture facilitates end-to-end traceability between design requirements, design decisions, and verification and validation (V&V) strategies [94]. This traceability is critical to safety assurance processes, but it is seldom considered in this context early in design, as will be described in Section 2.3.

A common architecting approach is to reason about a system in terms of (1) its *form* (what it is) and its *function* (what it does), (2) the entities that make up the system and their forms and functions, (3) the relationships between those entities, and (4) the emergent properties that result from those interactions [95]. From this perspective, a system architect "maps forms and functions".

Four common techniques help manage the complexity of the architecture [96]. First, *decomposition* breaks up entities into smaller entities to reduce the scope of consideration in the analysis. Second, *abstraction* hides internal complexity and maintains the form and function

necessary for a given level of analysis. Third, *hierarchy* helps organize entities at different levels. And fourth, *concept* creates the notion needed to guide the eventual architecture [96].

Several methods are commonly used to model system architectures. Model-Based Systems Engineering tools, such as the System Modeling Language (SysML) [97], are intended to link the architecture to other systems engineering artifacts using a common model instead of different documents. Numerical methods such as Dependency Structure Matrices (DSM) [98] and Network Theory [99] can help organize and quantify interactions between entities in an architecture. Finally, architectures modeled using the Department of Defense Architecture Framework (DoDAF) provide different views of the system to facilitate design from different engineering, operational, and managerial perspectives [100]. A variety of decision analysis and optimization methods to help steer architectural decisions are described by Crawley et al. in [98].

Unfortunately, there are two significant limitations to just using these approaches in the design of modern complex systems. First, to go from high-level requirements defined in natural language to a physical and logical architecture that informs detailed design activities requires a significant cognitive leap. Even though hierarchy, abstraction, and concept are supposed to help, the process of early decomposition often leads engineers to enumerate components before important functions of the system are identified. In practice, this often results in them falling back on previously developed architectures and their components to seed the design [101]. By doing so, they do not consider potentially better architectures to solve a problem.

Second, these modeling techniques overemphasize the *objects* of the system. However, in *control*-oriented systems, such as the novel aerospace systems of interest in this research, the representation of control becomes diluted among the objects, difficult to trace, and therefore nearly impossible to properly validate [101].

Leveson argues these limitations can be overcome by initially developing a conceptual architecture focused on top-down control interactions in a system [101]. Systematic analysis of this model can help identify and trace system-level requirements early to enforce desired emergent properties top-down and can then improve the use of conventional architecting techniques for more detailed design.

The above fundamentals and inspiration from several literature sources [39], [58], [98], [101] help scope the following definition for a system architecture, which is applied in this work:

> *At a higher level of abstraction, the architecture consists of the interactions supported by a control structure to determine the behavior of the collective system in the context of a broader system and environment. At a lower level, it consists of the components, their interactions, and their supporting control structure to determine how they contribute to the collective behavior.*

The remainder of this section surveys approaches that have been employed to identify functions the team must perform and methods to allocate these functions to collaborative entities within the team architecture.

2.2.2 Functional Analysis

Functional analysis is a process to identify the set of tasks to be performed by a collective system. It is often used in architecture design to allocate the function of those tasks to the system components (see Section 2.2.3). It is also a common input to various assurance analyses (see Section 2.3). This section explores existing functional analysis methods applicable to collaborative systems, including those used to analyze flight crew interactions, and how they are being adapted to explore human-machine teaming in novel aerospace systems like Urban Air Mobility (UAM).

Many functional analyses organize tasks in hierarchical structures to manage complexity. For example, Hierarchical Task Analysis (HTA) treats tasks as goal-specified behaviors that are attained through actions and feedback [102]. Tasks are identified through a systematic decomposition of goals and subgoals and are then used as input for other analysis and design techniques [2], [103].

Critical Task Analysis (CTA) employs this type of hierarchal task decomposition to analyze military flight crew responsibilities during mission execution [104], [105]. The process begins with identifying a limited set of high-level functions performed by pilots, such as *Aviate*, *Navigate*, *Communicate*, and *Manage* [106]. These are decomposed into activities that have defined time horizons, then into tasks needed complete the activities, and finally into subtasks as the primitive elements of execution. CTA supports behavioral, cognitive, information requirements, safety, and failure and degraded modes analyses used in military aircraft certification [105].

Civil aviation employs a similar functional analysis method in the advanced qualification program for air carrier flight crews operating under 14 Code of Federal Regulation (CFR) Part 121 and 14 CFR 135 [107], [108]. The Job Task Analysis (JTA) breaks up a mission into flight segments, similar to those in the military CTA, and hierarchically decomposes job functions into tasks, subtasks, and elements associated with each segment. Elements identify the knowledge, cognitive skills, motor skills, and attitudes required by humans to execute the subtasks. The output forms a basis for Crew Resource Management training curriculums.

Task decompositions based on existing aviation architectures are often used in the functional analysis of architectures for new aviation operations, like UAM. For example, NASA recently modified an existing JTA to identify low-level tasks potentially applicable to future UAM operators [109]. It then used a bottom-up approach to regroup tasks into higher-level categories to seed future human-automation functional allocation research. Another NASA report proposed a different functional decomposition of UAM functions akin to existing CTAs to demonstrate how to analyze safety using the guidance in Aviation Recommended Practice ARP-4761 [110].

There are other similar examples. The General Aviation Manufacturing Association (GAMA) developed a list of 13 “skill categories” based on a review of 14 CFR 61, with the intent to shift the responsibility for some of these skills away from the human and over to the automation [10]. NASA, in partnership with the FAA, also proposed 13 draft high-level functions, called Mission Task Elements (MTE), considered as fundamental building blocks to identify the relationship between aircraft performance, flight characteristics, and means of demonstrating compliance with future certification requirements [111]. Finally, the MITRE Corporation consulted with pilots to assemble a list of functions as part of its Behavior Competency Model (BCM), which explores safety assurance approaches for highly automated aviation applications [112].

There are significant drawbacks to these functional analysis methods. First, the process of decomposition reduces the analysis to focus on one function at a time and therefore does not consider emerging effects associated with interactions between functions. For example, the separate "skill categories" of *Landing* and *Emergency Procedures* proposed by GAMA [10] cannot be fully decoupled as there will be *Emergency Landings*.

In addition, the decomposition according to existing aviation operations implicitly guides future design decisions to mimic past architectures, and it does not consider potentially better solutions to address challenges in these new aviation operations. This may lead to similar supervisory control architectures, which face inherent limitations given the "automation conundrum" previously discussed in Section 2.1 [39], [44]. As described in the previous section, this shortsightedness is common in system architecting [101].

Alternative functional analysis methods have been proposed for new architectures. For example, NASA employed a top-down approach to systematically identify UAM aircraft control tasks for human-automation functional allocation research [113]. The process starts by decomposing high-level functions like *Mission Management*, *Flightpath Management*, *Tactical Operations*, and *Vehicle Control* associated with different flight segments. Next, subfunctions are identified for each by considering each of the four attributes of Resilience Engineering, which include *Monitor*, *Respond*, *Learn*, and *Anticipate* obtained from Hollnagel [114]. This novel method could be reframed to focus on abstraction-refinement and principles of system safety [38].

In Cognitive Work Analysis (CWA), tasks are organized hierarchically using a means-end abstraction of constraints and information requirements [115]. Each level of abstraction represents a different view of the overall work and explains *what* tasks are to be completed. The level above explains *why* those tasks need to be completed, or in other words, what higher-level *ends* they address. The level below explains *how* tasks need to be completed by describing children subtasks. This avoids some of the pitfalls of pure decomposition by acknowledging that functions at each level interact with multiple higher-level functions.

Pritchett et al. use this technique to model aircraft control in a functional allocation study described in the next section (see Figure 2-3) [116]. This arrangement, derived from the work of Rasmussen et al. [117], is well suited for systems theoretic analysis, as well as mapping resulting requirements in the form of an Intent Specification [52].

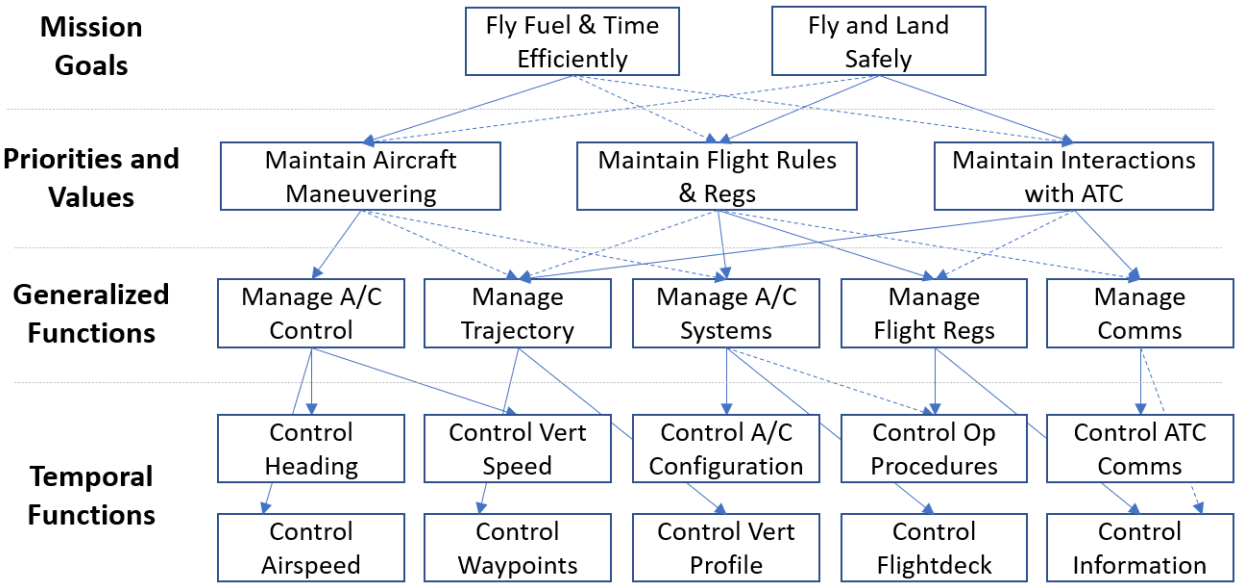


Figure 2-3: Means-Ends Functional Abstraction of Aircraft Control (recreated from [116])

The key takeaways from this survey of functional analysis methods are the following. First, most methods turn to analytical decomposition to outline functions to be performed by the team. This reductionist approach does not consider interactions between functions, and therefore it cannot help properly analyze emergent properties of the system, such as safety. Second, functional identification is often based on a bottom-up analysis of old architectures, which hinders the exploration of novel options potentially better suited for a given application. Third, there are elements of newer analysis methods that may help overcome some of these limitations. However, they are not currently implemented within a framework to analyze system safety and require further work to be useful toward the research objectives of this work.

2.2.3 Architecture Design Approaches for Collaborative Systems

Once designers have identified the system functions using methods like those described above, the next step in architecture design typically consists of allocating those functions to the system components. When doing so, engineers must also ensure that the system structure, such as its communication and control channels, can support that allocation.

The importance of functional allocation is emphasized in the literature. It is the earliest Human Factors design decision that can be made in a human-operated system [118]. It is a key property of human team architectures [58] and informs early decisions that later affect design, training, policies, and procedures [45], [46], [118]. Functional allocation, of course, also influences the safety of the system. When accidents occur with complex aerospace systems, it is nearly impossible to separate human actions from design flaws. It is therefore imperative to avoid *human-task mismatches* in these architectural decisions, as those often get mislabeled as *human error* in accident investigations [4].

Despite its importance, the proper way to allocate functions in collaborative systems remains open, as emphasized by a recent NATO working group [41]. When designing interdependent systems, there is a need to translate high-level concepts like teamwork and cooperation into the

implementation of control, interface, and behaviors of the system [2]. Some important considerations listed in the literature are to determine (1) if a problem benefits from collaboration between entities, (2) which tasks to delegate to each and when, and (3) to evaluate whether the design outcome meets system performance expectations [119].

However, the reality is even more complex than this for two key reasons. First, the design of these systems spans multiple technical disciplines, which in practice often work independently or in sequence from one another without effective collaboration. This is particularly challenging in the design of human-automation interactions (HAI), which requires integrated expertise in human factors and autonomy. Second, the designed systems are not closed. They interact with other systems and the environment, and it is the higher-level socio-technical system that must be safe. Little guidance is currently provided to engineers on how to make systematic architectural decisions with these considerations [39].

The rest of this section presents general approaches that have been employed to guide functional allocation architectural decisions for interdependent collaborative systems. Some of the common shortfalls encountered in these methods are discussed. Next, key methods are described in greater detail to showcase the state-of-the-art in this domain. While many of these tools were developed to help HAI design decisions, they are generally applicable to systems that seek the complex interdependent and collaborative interactions described in this work.

General Design Approaches

Very few structured approaches exist to design collaborative interactions in systems. At best, designers are equipped with general systems engineering practices and separate highly specialized and siloed domains to integrate. The disparate natures of disciplines, like human factors and autonomy, make this particularly challenging. To bridge this gap, designers often turn to high-level design patterns to guide functional allocation architecture decisions.

One of the earliest frameworks for functional allocation between humans and machines, proposed by Fitts in 1951, is called *Men Are Better At – Machines Are Better At* (MABA-MABA) [66]. In this approach, decomposed functions are individually assigned in a binary fashion to either the human or the autonomy based on a lookup table listing their relative strengths for different types of tasks. The simplicity of this method has kept it attractive over the years.

However, MABA-MABA has also received wide criticism that the approach does not consider the emerging effects of the allocation. For example, naively assigning taskwork to the automation does not necessarily reduce human operator workload, but instead, it transforms its nature [120]. It often creates new higher-level tasks for the operator to direct, monitor, and adjust the automation in its execution. Another side effect of this reductionist approach is that no consideration is given to the holistic set of tasks assigned to each entity. In practice, this results in attempting to automate as much as possible and leaves a non-cohesive patchwork of tasks for the human to deal with [4]. Finally, this oversimplified framework provides no avenue to promote collaborative task work.

More recently, Crouser et al. proposed a modern take on the MABA-MABA lookup tables by reframing them to emphasize collaborative work [119], [121]. They distill an extensive survey of human-computer collaborative systems into a set of *affordances*, described as partnership opportunities for collaborative action. While this framework does shift emphasis away from

binary allocation and toward collaboration, it is still limited to a set of design patterns to apply to decomposed tasks that provide little systematic and holistic system design guidance.

Another popular design pattern called *Levels of Automation* (LOA) was developed by Sheridan and Verplank in the late 1970s through the study of human supervisory control of automated systems. The first version of the LOA framework consists of a scale of 1 to 10, where 1 represents full human manual control with no computer assistance, and 10 is fully autonomous with no human involvement (see Table 2-1) [122]. The LOA is often used to set the tone on the general level of authority provided to the automation in the design of a system. Since, multiple variants of this taxonomy have been proposed [70], [111], [123]. A 1 to 5 scale version of LOA developed by the Society of Automotive Engineers (SAE) has come to dominate the design of "self-driving" cars [124].

Table 2-1: Levels of Automation (based on [122])

Level	Description
1	Human does it all
2	Computer offers alternatives
3	Computer narrows alternatives down to a few
4	Computer suggests a recommended alternative
5	Computer executes alternative if human approves
6	Computer executes alternative; human can veto
7	Computer executes alternative and informs human
8	Computer executes selected alternative and informs human only if asked
9	Computer executes selected alternative and informs human only if it decides to
10	Computer acts entirely autonomously

The same simplicity that has made the LOA framework so popular is also often criticized in the literature. LOAs explicitly describe machine capabilities alone and not those of humans or other machines working with it [39]. LOAs do not facilitate a collaborative behavior [120], they are too coarse to be useful given a complex set of tasks to be allocated within a team [116], and they do not address authority-responsibility mismatch where the controller is ultimately accountable for the successful outcome of the task is different from the one performing it [125]. The National Academy of Sciences concluded: "The application of autonomy concepts and technology to a system is inherently a complex issue, with several degrees of freedom that must be addressed. Thus, it is impossible to characterize the implemented degree of autonomy completely with a single number" [47].

To provide additional flexibility in functional allocation, one popular version of the LOA taxonomy decomposes each function into four subfunctions: (1) information acquisition, (2) information analysis, (3) decision and action selection, and (4) action implementation [126]. This framework acknowledges that each function performed by the system can be implemented at a different LOA, and its associated subfunctions can vary in LOA too.

This type of approach has invigorated calls to continue updating the Levels of Automation frameworks with additional details, including the metric of *Satisficing* in which the human selects an allocation that is "good enough" [123]. However, even if more detailed LOAs help designers

cognitively reason with a design concept, they alone still do not provide systematic guidance on how to assess architectural decisions holistically.

In addition to these high-level frameworks, the literature also offers multiple sets of design best practices for teaming systems. Endsley presents a table of design guidelines for Human-Autonomy systems [44], which provides guidance like "automate only if necessary – avoid out-of-the-loop problems if possible". Similar lists include the ten "Human-Autonomy design first principles" by Mosier et al. [23], five "Human-Autonomy key design principles" enumerated by Ho et al. [14], and "Human-Autonomy Teaming design tenets" described by Battiste and Shively [5], [13].

While these guidelines can be useful for general reference, they do not provide systematic guidance to inform design decisions for complex systems. However, more specific collaborative architecture design strategies are also found in the literature, as described below.

Specific Design Approaches

One approach by Miller & Parasuraman uses a means-end hierarchical decomposition of the team activity to formulate a set of "plays" that allow operators to efficiently delegate tasks to autonomous systems [127]. Instead of forcing the system engineer to perform a final functional allocation during design, this approach lets the operator make flexible allocation decisions during operation. Tasks can be delegated at high or low LOAs, tailoring how involved the human supervisor wants to be in planning and execution details, much like what is done in human supervision [127]. Unfortunately, no guidance is provided to design "plays", so the detailed implementation and how it is analyzed for safety or other properties are not addressed.

Another framework developed by Heisey et al. helps systematically identify system requirements for teams of unmanned systems and traces them from Concept of Operations, through architectural design, and to V&V [89]. The process employs organized Subject Matter Expert interview templates to define team-level architectural requirements, which then feed lower individual system entity requirements for implementation, and then validation. The process is top-down and has feedback loops between each level. However, it provides no systematic way of exploring emerging properties of design decisions, other than implementing them into a simulator to observe the behavior of the system. This method of validation is inherently limited, as will be described in the next section.

A method by Dearden et al. [46] allocates human and automation functions by first treating the team as a black box to identify high-level functions it must perform in the context of an operational scenario. Some functions are initially allocated based on physical constraints (e.g., automation must be used for low-level control of an aerodynamically unstable aircraft) or regulations (e.g., decisions to release weapons must be made by humans). Next, candidate functions for total and partial automation are evaluated for implementation feasibility. Coordination tasks between the human and automation that emerge from the selected functional allocation are identified and fed back as new functions for additional iterations of the design process. Finally, the design decisions for different scenarios are compared to consider global tradeoffs, and changes are fed back into the process for more interactions [46].

The key strengths of Dearden's method are its ability to maintain a system-level view and to produce systematic design decisions that produce traceability and rationale. However, there are

several concerns. First, as acknowledged in his paper, the framework is rigidly defined between a single operator and automation, and it must be expanded to handle more general system compositions. Second, there is no clear process to move up or down in abstraction to analyze different levels of details. Finally, this leads to scalability concerns for complex systems that would require a large number of these workload-intensive algorithmic iterations.

Coactive Design, by Johnson et al., helps design systems with any number of interdependent humans and/or machines [2]. It emphasizes three key points. First, components have various capacities (e.g., knowledge, information, skills, and abilities) that define how well they can perform certain tasks or support teammates. Second, teammates can help close each other's control loops. For instance, a robot can sense obstacles, but a human can help interpret how to interact with them. This requires teammates to be able to *Observe*, *Predict*, and *Direct* (OPD) each other, as shown in Figure 2-4. Finally, design decisions relating to these interactions are evaluated in the context of the other decisions made for the system, similar to Dearden's method above [46].

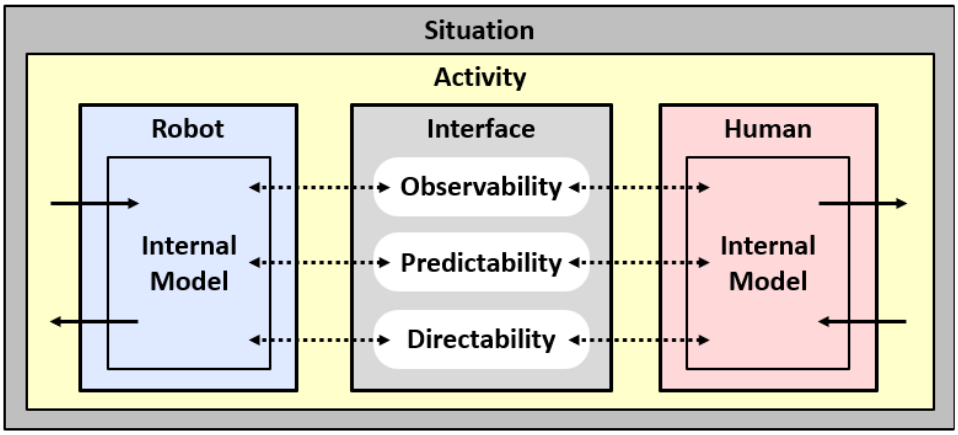


Figure 2-4: Coactive System Model (adapted from [2])

The process, shown in Figure 2-5, begins with a functional analysis using HTA decomposition (see Section 2.2.2). For each task, each teammate is enumerated as having a *primary* and a *supporting* role. Capacities required to support those roles are listed, beneficial interdependent relationships are identified, and OPD requirements are generated to support those relationships. After solutions are designed to meet these requirements, those designs are analyzed to see how they impact the other relationships in the system. Any changes that occur in those relationships are fed back into the process for iteration until the design converges with no changes.

The literature identifies Coactive Design as the state-of-the-art in designing interdependent systems [23], [39]. However, a weakness of the approach is that the initial task decomposition leads to a bottom-up approach to design, which is less effective than top-down approaches in proactively enforcing emergent properties like safety [38]. Similar to Dearden's method [46], this approach does not offer a path to navigate between different levels of abstraction and is difficult to scale for large complex systems. Finally, it does not explicitly consider interactions between the interdependent entities and the higher-level socio-technical system it integrates with.

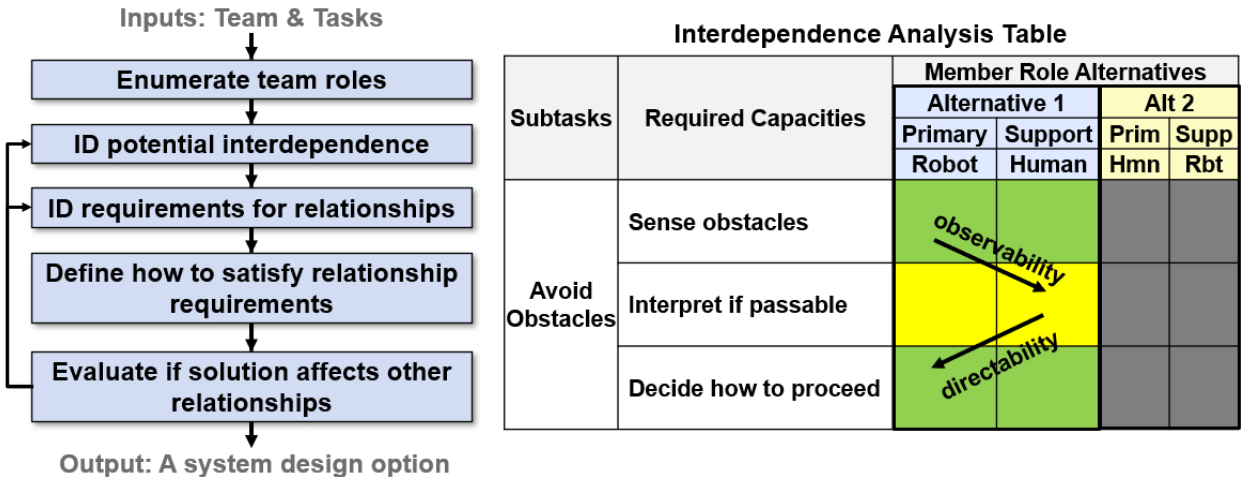


Figure 2-5: Coactive Design Process (derived from [2])

One final architecture design approach, by Pritchett, Feigh, and Kim explores functional allocation using computational simulation of collaborative work [116], [118], [125]. Their method starts with a means-end decomposition of the collective taskwork, which is then allocated to the teammates in different ways to simulate the joint activity. If an entity is allocated a task for which another teammate is responsible, then monitoring tasks are created for that teammate as part of the emergent teamwork in the activity.

The simulations are bounded by formal requirements that must be met by the architecture design and help observe the performance and resulting total work involved in each functional allocation trial. The results of these simulations are used to develop parameters for a network representation of the team architecture to optimize functional allocation [99].

The key strengths of Pritchett's work are the formal requirements and metrics specified to support architectural decisions and the ability to assess emergence in teamwork based on those decisions. However, the use of simulation alone is not sufficient to produce a robust verification and validation of the design, as discussed in the next section. In addition, a global optimization scheme like this ultimately boils down to a decomposed set of cost function parameters, which are difficult to define objectively [46].

2.3 Safety Assurance of Teaming Systems

Safety Assurance refers to the set of activities taken to provide confidence that system hazards have been eliminated or controlled [4]. This research focuses on the subset of activities most relevant to initially field an aerospace system: hazard analysis, verification and validation (V&V), and certification. As identified in Chapter 1.2, the literature recognizes there are significant challenges associated with these processes, especially for systems with complex interdependent and collaborative interactions.

Typical assurance processes are both applied too late and are inadequate [4], [38]. Unfortunately, safety assurance is often applied as a separate discipline from system design and only becomes emphasized in later development stages, on the right side of the Systems

Engineering "V" (Figure 2-2). This practice hinders the ability to build safety into the system from the beginning. Instead, it often results in only having less effective and more expensive design change options to address safety problems that are recognized later in the lifecycle.

In addition, existing processes are unable to handle complex systems holistically and must therefore conduct separate assurance methods for hardware, software, and humans. The collaborative and interdependent interactions sought in the novel aerospace systems described in Chapter 1 exacerbate these issues. This is especially true when human and software control is highly coupled.

The following is a discussion of typical methods to conduct hazard analysis, V&V, and certification in aerospace systems. In addition, it addresses some of the recent attempts to tailor these processes to handle novel aerospace systems with complex, team-inspired, component interactions.

2.3.1 Hazard Analysis

Hazard analysis is a process to systematically identify causal factors that can lead a system to enter a hazardous state. Its output is the foundation for other assurance processes, including definition of system requirements and constraints to enforce safety, implementation of these requirements, V&V, and documenting results for certification. For this reason, hazard analysis should begin influencing system design early in its conceptual stages, on the left side of the Systems Engineering "V". However, in practice, this does not happen until much later [4].

There are different types of hazard analyses, but they all consist of the following four basic functions [4]. First, they specify the system losses that are unacceptable to the stakeholders. Next, they define the scope of the system analyzed, which typically is defined by the system boundary, over which designers have engineering control. Third, they identify hazards, which are system states or sets of conditions that, together with a particular set of worst-case environmental conditions, will lead to a loss [50]. Finally, they help find causal factors that can lead to these hazards. Results can then be used to establish design recommendations to eliminate or mitigate the hazards.

There are numerous hazard analysis techniques, and some of the most common ones are described in detail by Ericson [128]. These techniques are founded on *causality models*, which impose patterns on observed events and represent assumptions about how the world operates [4]. In safety engineering, these models help explain why accidents occur. Leveson has categorized causality models into four types: *Energy*, *Epidemiological, Linear Chain of Events*, and *System-Theoretic* [4]. Brief descriptions of techniques associated with these models follow.

Energy Models

Energy models assume accidents are caused by an uncontrolled and undesired flow of energy from a source to an object at-risk [4]. Sources of energy that may be considered include kinetic, radiation, chemical, thermal, electrical, acoustic, or biological. Hazard analysis techniques built on these models typically consider flow control mechanisms, such as barriers and alerts on the state of the flows help mitigate hazards [128].

This type of simple analysis can be used to identify hazards to be further explored using other techniques [129]. However, its limited scope is not adequate by itself to address complex control-oriented systems like those found in advanced aerospace concepts.

Epidemiological Models

Epidemiological models treat accidents as a public health problem by conceptualizing them in terms of an agent (physical energy), the environment, and a host (victim). Accidents are explained as resulting from complex and random interactions between these three factors and cannot be rationalized by considering only one of them or by a simple sequence of events [4]. The statistical reasoning used in public health problems is then applied to analyze accidents.

Unfortunately, there is no known hazard analysis technique built onto epidemiological models [4]. In addition, complex systems are too structured to be analyzed using the same statistical processes applied to large populations [130]. Furthermore, the quality of the records of designed system anomalies may be too limited for this type of statistical analysis. These reasons limit the use of epidemiological models in safety assurance activities.

Linear Chain of Events Models

The majority of hazard analysis techniques used in traditional safety assurance programs are based on *Linear Chain of Events* causality models. Linear causality implies that if variable *A* has a causal influence on *B*, then *B* has no influence on *A*. While this assumption makes it easier to identify the "chains of failures" often reported in accident reports, it is an oversimplification that actually hinders effective safety engineering [38]. The following are the associated techniques most commonly used in aerospace safety analysis.

Functional Hazard Analysis (FHA) is an inductive and qualitative technique in which system functions are systematically decomposed, and then individually analyzed for how they could fail. The analysis documents the failed function, its associated hazard, causal factors, mitigation recommendations, and assessed risks [128]. Civil aviation practices recommend conducting an FHA for qualitative analysis early in design to then feed follow-on quantitative analyses later in the engineering lifecycle [131].

Failure Modes and Effects Analysis (FMEA) follows similar inductive reasoning as FHA. The analyzed system is decomposed into components. Next, the consequence associated with the failure of each of these components is evaluated using a bottom-up approach. This analysis is often expanded quantitatively by including failure rates for the components using service experience, accelerated testing, or industry standards [132]. Because FMEA considers all component failures and not just those that lead to hazards, it focuses more on reliability than safety [128].

Failure Modes and Effects Criticality Analysis (FMECA) is a more detailed version of the FMEA. It includes an assessment of the criticality of component failures and considers mechanisms to detect them [128]. FMECA is commonly applied to military aviation systems [133].

Fault Tree Analysis (FTA) uses deductive reasoning by starting with a hazard, and systematically searching for potential chains of events that can lead to it. The relationships in

these sequences of events are described using Boolean logic. FTAs are often used quantitatively, by similarly adding failure rates of components, and computing the overall probability of any given event chain [128].

There are major limitations associated with these hazard analysis techniques. They all focus on component failures and do not consider how the interactions between fully functional components can lead to hazardous states [38]. The assumption that causality is linear is a significant weakness recognized by the community since cyclic relations are common in real systems [134], [135]. All of these methods are hardware-oriented, and none can effectively handle the contributions of human and software control to accidents [38].

The quantitative aspects of these methods are also problematic. Failure rates cannot be accurately determined except for standard hardware components and designs with extensive historical use. These numbers cannot be incorporated until sufficient design detail is available, which limits the ability to analyze and influence early design concepts [4]. Furthermore, failure rates are unknown for software, sophisticated human behaviors, and new technologies and designs.

Quantitative assessments must also often rely on flawed or oversimplified assumptions, such as probabilistic independence between events [135]. Some techniques do exist to evaluate common cause failures in FTAs. However, those methods are challenging to execute rigorously [128], and they also often rely on their own superficial common failure rate assumptions [136]. As a result, probabilistic assessments are usually inaccurate, and they can even be misleading in characterizing hazards.

Despite their limitations, these hazard analysis techniques are precisely the ones that are called for in aerospace system safety standards, such as ARP-4761 [131] for civil aviation and MIL-STD-882E [133] for the military. These standards acknowledge some of the challenges in applying these techniques to software components, and they only superficially address the role of human control. Separate systems engineering standards, such as DO-178C [137] and MIL-STD-46885A [104], provide some safety considerations for software and humans respectively, but they lack actual hazard analysis techniques. This patchwork creates a disjointed approach to aviation system safety, which is ill-suited to handle the complexity of modern systems [4], [105].

Hazards and Operability Analysis (HAZOP) is another technique based on a linear chain of events model. It is not called out in the aviation standards listed above, but it has been employed in modern aerospace systems, such as the Airborne Collision Avoidance System [138]. HAZOP overcomes some of the limitations by using guidewords to systematize a search of causal factors in a physical design specification. It encourages multi-disciplinary qualitative reasoning and considers deviations from operating intentions that go beyond failures [4].

However, HAZOP also shares some of the weaknesses found in the other techniques. These include the assumption of linear causality, the focus on physical objects instead of software control, and the inability to apply it at the earliest stages of design. In addition, this method, much like the others, is often criticized for being labor-intensive, time-consuming, and difficult to review [4].

System Theoretic Models

System Theoretic models, which are generally newer than chain-of-event techniques, break away from the assumption that causality is linear [4]. They acknowledge causal loops exist between components, where variable *A* can influence *B*, and *B* can also influence *A*.

System-Theoretic Accident Model and Processes (STAMP) is one such causality model, which treats safety as a control problem rather than a failure problem [38]. STAMP provides a unified framework to look at how component interactions between hardware, software, and humans can lead to unsafe system behaviors.

System-Theoretic Process Analysis (STPA) is a qualitative hazard analysis technique built upon the STAMP causality model [50]. Its foundation in Systems Theory allows it to overcome many of the limitations of other methods. For this reason, this technique is becoming increasingly popular in many industries, including aerospace.

Systems Theory, STAMP, and STPA provide the foundation for how safety will be analyzed in this work. They are described in greater detail in Section 2.4, along with an argument for their selection.

Hazard Analysis in Systems with Team-Inspired Interactions

There is an overwhelming recognition that new aerospace systems with interaction inspired by teams face significant challenges in V&V and certification (see Chapter 1.2). Despite this, the literature is relatively silent regarding the limitations associated with hazard analysis techniques for these systems. This is surprising given the critical role of hazard analysis in the other safety assurance activities.

NASA demonstrated how to employ methods like FHA and STPA in new aviation applications like UAM and is still exploring different methods to best address these novel technologies [110], [139]. Belcastro et al. describe a need to evolve hazard analysis techniques to address distributed multi-UAS operations. However, their analysis is limited to the study of historical data to identify potential hazards for the analysis of future systems [140]. Baig et al. survey applications of FTAs and highlight how they fall short in human-machine interactions, leading them to use Fuzzy Logic or other augmentation mechanisms to account for the non-deterministic nature of humans [141].

Finally, a growing body of research using STAMP-based techniques and STPA, described in Section 2.4, demonstrates how to apply these methods to aerospace systems with complex interactions. However, it also illustrates some of the challenges that must be overcome so that these interactions can be rigorously analyzed.

Simply put, there is a clear gap in the literature in addressing hazard analysis specifically for systems that exhibit complex interactions, such as those inspired by human teams.

2.3.2 Verification & Validation (V&V)

Verification and Validation (V&V) is a broad term to describe a set of activities that take place throughout the development lifecycle, but that are most often emphasized on the right side of the

Systems Engineering "V". Verification determines if the system meets its design requirements, and is commonly referred to answering "Did you build the thing right?". Validation assesses if the planned system meets the user's operational needs, and answers "Did you build the right thing?".

In the context of safety assurance, V&V determines if causal factors identified in the hazard analysis have been eliminated or mitigated. This determination is usually based on the results of simulations, testing, and formal mathematical reasoning [4]. Each method has strengths and limitations described below.

Simulation and Testing

Simulation and testing are the most common tools used in V&V. They can be effective at evaluating design decisions, characterizing system performance, and finding design flaws. However, there are many examples in the teaming literature where the hazard analysis is bypassed, and causal factors are identified using only simulation and testing [5], [89], [91], [116], [142].

The hope with this strategy is to address discrepancies found in simulation by modifying design requirements, implementing them, confirming their success in additional simulations, and then gaining confidence in the system behavior through live testing. While this practice can be helpful in early concept exploration, it is inadequate if those are the only assurance steps performed to field safety-critical applications. Simulations only reveal what they were intended to simulate and rely on assumptions and simplifications in the dynamics of the system and its environment [143].

Testing is typically expensive and cannot be performed exhaustively to cover the large number of states achievable by complex systems in complex environments, especially for software-intensive systems [144]. Testing may reveal the presence of a problem but cannot show the absence of it[3]. In addition, systems are tested against requirements, but that does not prevent the requirements from being incomplete with respect to safety. In fact, most accidents can be traced back to incomplete or otherwise flawed requirements [4].

Formal Methods

The costs and limitations associated with testing have led to a growing interest in using *Formal Methods* (FM) to support assurance in novel aviation systems. Formal Verification involves applying inputs to a system to check if all its executions meet its requirements set *R* [150]. Either a counter-example is found that shows *R* can be violated, or the algorithm provides proof that the design is correct. The verification is *sound* if its proof is valid and all behaviors meet *R*, or if the counter-example is an actual behavior of the system that violates *R*. It is *complete* if it provides a solution for all inputs to the system [150].

New aviation certification standards developed to address software and automation define and encourage the use of FM, including DO-333 [151], ASTM-TR1 [111], and ASTM-F3269 [152].

---

[3] The author has observed this lesson many times over years of developing and flight-testing complex multi-UAS systems under distributed control [145]–[149].

Several aviation software systems have employed formal verification on portions of their software using techniques such as *Model Checking*, *Theorem Proving*, and *Abstract Interpretation* [153], [154].

Formal methods have been employed in applications relevant to teaming. A large body of Human-Automation Interactions (HAI) studies aim to prove the correctness of HAI interfaces [155], [156]. The systems are often modeled as finite-state transition systems, and temporal logic is used to check (1) the reachability of states, (2) the visibility of visual feedback as a result of actions, (3) task-related properties to support the human, and (4) the consistency of the software implementation with its requirements. These types of studies do not depend on a model of the human to ensure the interface behaves as intended [157].

However, other HAI research applications do attempt to create a "formal model" of the human operator. Studies apply FM to model operator mode confusion [157], operator activity sequences [158], humans motor capabilities [159], the effects of human control latency on HAI performance [160], and human-robot interactions when in close proximity to one another [161]. Unfortunately, the concept of a formal mathematical model to describe the complex, adaptive, and often unpredictable nature of human behavior is inherently limited and presumptive. While this practice can explore design states to help designers identify areas that need improvement or to select follow-on candidate test scenarios with real humans, as exemplified by Bolton et al. [158], formal guarantees derived uniquely using FM in this context may be misleading.

Three different applications illustrate the state-of-the-art in applying FM to aerospace systems with collaborative component interactions. First, formal models of teamwork have been used to guide functional allocation design decisions in aviation human teams and human-machine teams. For example, interactions between pilots and air traffic controllers were modeled as a task tree with temporal relationships to formally verify the behavior of aircraft heading changes [162]. A similar formulation, previously discussed in Section 2.2, formally verified that aircraft control functional allocations between humans and automation met specified requirements [116], [118], [125].

In a second application, FM are employed to formalize aircraft control activities and decisions made by human pilots to shift some of those responsibilities to automated systems. For instance, a body of NASA research formalizes how human pilots *see and avoid* other aircraft. This has resulted in the development and formal verification of several detect and avoid algorithms for UAS or automated copilots [163], [164]. However, this research also illustrates challenges in creating valid formal models of these complex systems [165]. In addition, it shows that unexpected behaviors not identified by the formal methods can arise when the system interacts with a human pilot during flight testing [166].

Finally, formal verification has been applied in Run-Time Assurance (RTA) research. The hope of RTA is to provide a simpler and fully certified *pedigreed* controller to monitor a *non-pedigreed* complex function and, if necessary, recover to a safe state [152]. Proponents of RTA argue the protected system is equivalent in terms of safety to a fully verified system [90], [167]. Some also promote it as a mechanism to transfer responsibility from a human pilot over to automation in new aviation applications [78], [111], [168]. RTA has been studied to monitor control of a distributed multi-UAS system [90] and real-time monitoring has been actually fielded on fighter aircraft to protect pilots from colliding with terrain [169]. However, it is also noted that

the later fielded example was verified through flight testing, and no known RTA system has been operationalized using FM to date.

Despite the interest, FM face challenges that have kept their adoption rate low for assuring fielded systems. Some FM techniques, like theorem proving, require a high amount of expertise to execute and limit the pool of people that can successfully apply them [153], [158]. The low adoption also means few developers get exposed to how to use it [153]. Furthermore, this framework does not yet address some of the latest concepts in autonomous control that involve adaptive and non-deterministic functions like those popular in Machine Learning (ML) [150].

Formal techniques face other inherent limitations. The process of developing formal specifications from informally stated customer requirements introduces a risk of formally representing the wrong requirements [170], [171]. Ensuring model validity is challenging as the mathematical formulation of a problem is only an approximation of the real world and differs from implementation using programming languages [165], [167]. FM are difficult to scale up beyond simple problems and are limited in their ability to handle complex real-world systems [90], [158], [167], [170]–[172]. Finally, the environment the system interacts with is impossible to formally describe due to its unpredictable nature and unanticipated behaviors [170].

These challenges and limitations do not mean FM cannot play a role in safety assurance. But FM is not the solution to all problems, and it is just one of several tools available including hazard analysis, simulation, and testing.

2.3.3 Certification

Certification is a mandatory activity for safety-critical applications like aviation. It consists of providing the results of earlier assurance efforts, like system behavior specifications and their V&V, to a governing body to legally recognize that the system complies with requirements, including safety [153]. Certification requires demonstration, with appropriate and accessible substantiation, that (1) the system achieves its intended function, and (2) unsafe system behaviors are known, understood, and mitigated [110].

Certification bodies establish regulations to govern the certification process, such as 14 CFR Part 21 [49] for aircraft certification by the FAA or AR 70-62 [173] for the US Army. These regulations point to standards developed with industry, such as ARP-4761 [131], DO-178C [137], and MIL-STD-882E [133], which define recommended practices to follow to achieve certification. These standards all point to the hazard analysis and V&V techniques previously surveyed, and therefore they suffer from the same limitations previously described. However, the literature describes additional challenges in certifying teaming systems, as reviewed below.

One significant difficulty is that technologies enabling autonomous systems are so novel that they are not addressed by prescribed regulatory certification requirements [110]. For example, non-deterministic and adaptive technologies, like those found in ML, are not covered at all by existing regulations, and therefore, not one has been certified to date [39], [42], [150], [152].

For this reason, the FAA recently shifted from a prescriptive-based to a performance-based approach to certification. This offers flexibility to consider rapidly evolving novel technologies but also creates concerns. A performance-based approach does not encode past lessons learned like prescriptive methods do and does not provide any unified guidance to manufacturers and

regulators on a specific approach to demonstrate the system is safe [4]. This opens up the risk of making "Safety Cases" for certification, which tend to be based on informal arguments, are prone to confirmation bias and omission of information, and may be made under significant pressure to get a finished product out to market [174], [175].

Another challenge specific to aviation is that the regulatory framework is predicated on a human pilot in command having direct responsibility and final authority for the operation of an aircraft [49]. This creates a barrier in certifying novel human-machine collaborative control paradigms that shift some of these responsibilities onto autonomous systems. While this topic was previously raised for existing autonomous systems like flight envelop protection, it is still an open debate [3]. As a result, industry is advocating for regulatory changes to allow autonomy to become increasingly, if not fully, responsible for flight operations [176], [177].

Some preliminary certification frameworks have been proposed to reason about the division of functional responsibilities between multiple human and machine entities [111], [112]. They document V&V strategies based on functional allocation decisions, risk, and technology maturity. However, aside from providing basic heuristics, these frameworks lack rigor in their decision support, they do not address some of the fundamental aspects of team interactions described in Section 2.1, and their functional decomposition approach does not lend itself to assessing safety at the system level.

One final certification challenge to note is that current standards are vehicle-centric for insertion into a rigid aviation infrastructure and framework. Processes currently exist to certify Unmanned Aircraft System (UAS) operations on a case-by-case basis, which do not fit within the traditional aviation ecosystem. However, the resulting certifications are highly restrictive and narrowly scoped [178]. This approach increases the certification burden, and there is concern that it will create a barrier to entry for non-traditional aviation manufacturers in new aviation markets [179]. NASA is interested in creating a more holistic certification program, but no such framework is known to exist today [39], [110], [180].

2.4 Systems Theory and STAMP

The approach developed in this dissertation to model, analyze, and design the safety of systems with complex team-inspired interactions is grounded in Systems Theory and the System-Theoretic Accident Model and Processes (STAMP) causality model. This section provides a brief overview of this foundation. It highlights the strength of STAMP-based hazard analysis techniques compared to other methods as an argument to select it for this work. It also describes past research in STAMP related to teaming to motivate the need for analytical expansion.

2.4.1 Introduction to Systems Theory, STAMP, and STPA

A *system* is defined as "a set of components that act together as a whole to achieve some common goal, objective, or end" [1]. Systems exhibit emergent properties, such as safety, which arise from the interactions among these components. These properties can only be analyzed for a system as a whole and cannot be treated as a composition of individual component behaviors. Most

designed systems are considered *open* as they interact with the environment, which is the set of components and their properties that are not part of the system but can affect its state [1].

Systems Theory

Systems Theory was developed over the last half-century to augment the scientific method in studying increasingly complex and interactive systems developed by humans [181]. Prior to Systems Theory, analysts were limited to using analytical decomposition or statistical methods.

Unfortunately, modern systems can no longer be analyzed using only reductionist approaches. These systematically break a system down into smaller components and synthesize their individual behaviors to predict that of the system as a whole. Decomposition distorts the analysis of the collective behavior since components (1) do not operate independently, (2) do not behave the same when examined singly, (3) are subject to feedback loops, and (4) face more than simple pairwise interactions [1]. In addition, modern systems also have too much structure to behave randomly. As a result, they cannot be analyzed using statistical methods used to study much larger natural systems, like populations [130].

Systems Theory complements these other methods by focusing on the system as a whole rather than a conglomeration of parts. It emphasizes holism, interconnectedness, interdependence, context, dynamic complexity, and non-linear causality [181]. In Systems Theory, the term *system* is recursive in that systems are made of systems. Any level can represent a system, a system of subsystems, or a system of systems. These terms all represent the same thing, so there is no need to treat them differently [182].

Systems Theory is founded on two guiding principles: *Emergence and Hierarchy*, and *Communication and Control* [183]. The first principle means that systems are modeled using hierarchal levels, where each level is more complex than the level below. The second principle implies that controls can be applied from any level in the hierarchy onto the level below to constrain degrees of freedom on its emergent behavior. These two concepts are at the core of STAMP.

System-Theoretic Accident Model and Processes (STAMP)

STAMP is a relatively new causality model grounded in Systems Theory, which treats safety as a control problem rather than a failure problem [38]. It assumes that accidents are caused by unsafe component behaviors as well as unsafe interactions among system components. These components can be hardware, software, and human in nature. STAMP, as in Systems Theory, employs abstraction as a means to understand system complexity.

The three main parts of STAMP are Safety Constraints, Hierarchical Control Structures, and Process Models [38]. Constraints are imposed by any system level on the level below to enforce emergent properties and behaviors. Functions are modeled with control structures using feedback control loops between controllers and controlled processes at different hierarchal levels. Each controller may be controlled by a higher-level controller, each controlled process may be the controller of a lower-level entity, and feedback loops may exist between multiple different levels (see Figure 2-6). Feedback control loops are defined by four conditions grounded in Control Theory: goal condition(s), controllability, process model, and observability [38].

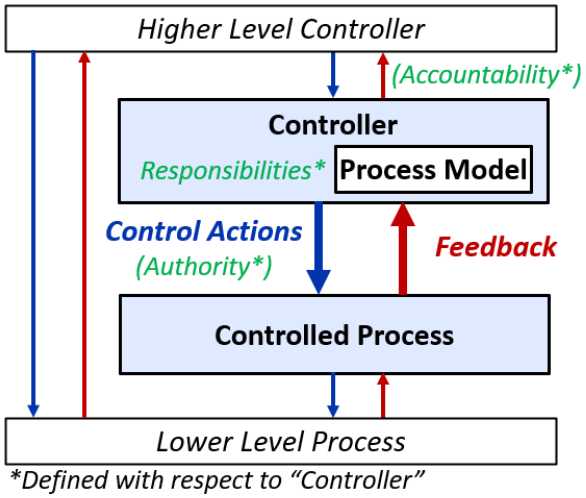


Figure 2-6: Example STAMP Hierarchical Control Structure

STAMP, and its analysis tools, such as STPA described in the next subsection, have recently gained popularity in many industries. In the past decade, an entire body of literature has emerged involving applications of STAMP, extensions, and comparisons of its tools with other methods [184].

System-Theoretic Process Analysis (STPA)

System-Theoretic Process Analysis (STPA) is a hazard analysis tool built with STAMP as its theoretical foundation. The analysis is conducted in four steps as described in the STPA Handbook [50] and shown in Figure 2-7. These are briefly described below with an illustrative example analyzing the safety of a multi-UAS system under human supervisory control described by Johnson, Kopeikin, & Leveson [143].

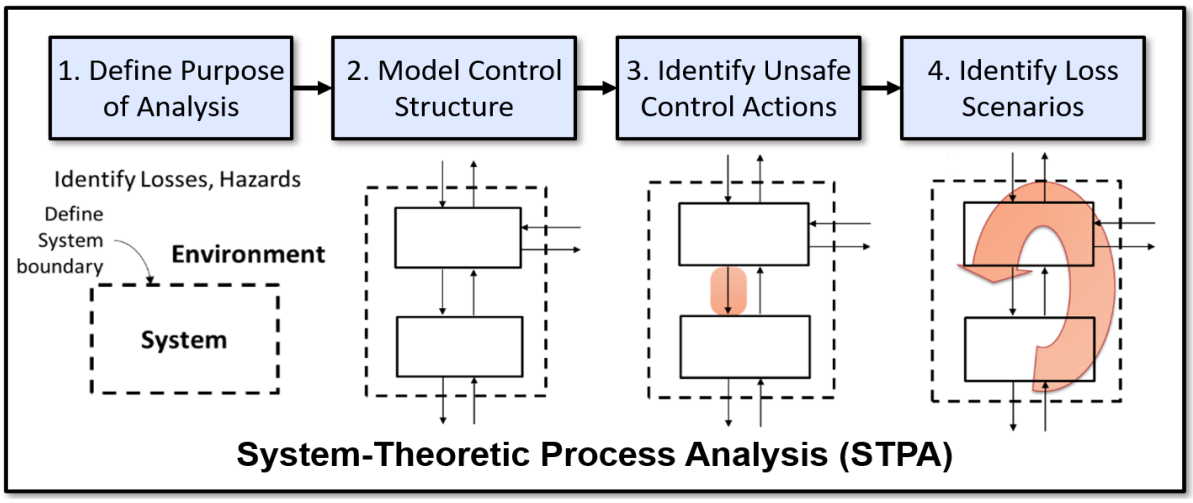


Figure 2-7: Four Steps of the System-Theoretic Process Analysis (STPA) [50]

Step 1 first defines the purpose of the analysis and the assumptions about the system and the environment. In the illustrative example, it is to analyze safety hazards for the multi-UAS system starting early in conceptual design.

Next, system *losses* that are unacceptable to the stakeholders are identified. For the multi-UAS system, these may include (L-1) loss of mission, (L-2) loss of life or permanent disabling injury, and (L-3) loss or damage to the UAS or equipment [143].

Finally, system-level *hazards* are defined and traced to the losses they can lead to, as shown in Table 2-2. A hazard is "a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to a loss" [50, p. 17]. System-level *constraints* can then be derived to eliminate or mitigate these hazards.

Table 2-2: STPA Example Multi-UAS System Hazards [143]

Hazard ID	Hazard Description	Loss Traceability
1	UAS does not complete mission objectives	L-1, L-2
2	Structural integrity of UAS is violated	L-3
3	UAS separation standards are violated	L-1, L-2, L-3

Step 2 models the system as a hierarchical control structure, in which responsibilities, control actions, and feedback elements are defined for each controller. It is encouraged to start the analysis at a high level of abstraction before considering adding additional details in refinement. Figure 2-8 shows the control structure for the multi-UAS system example [143].

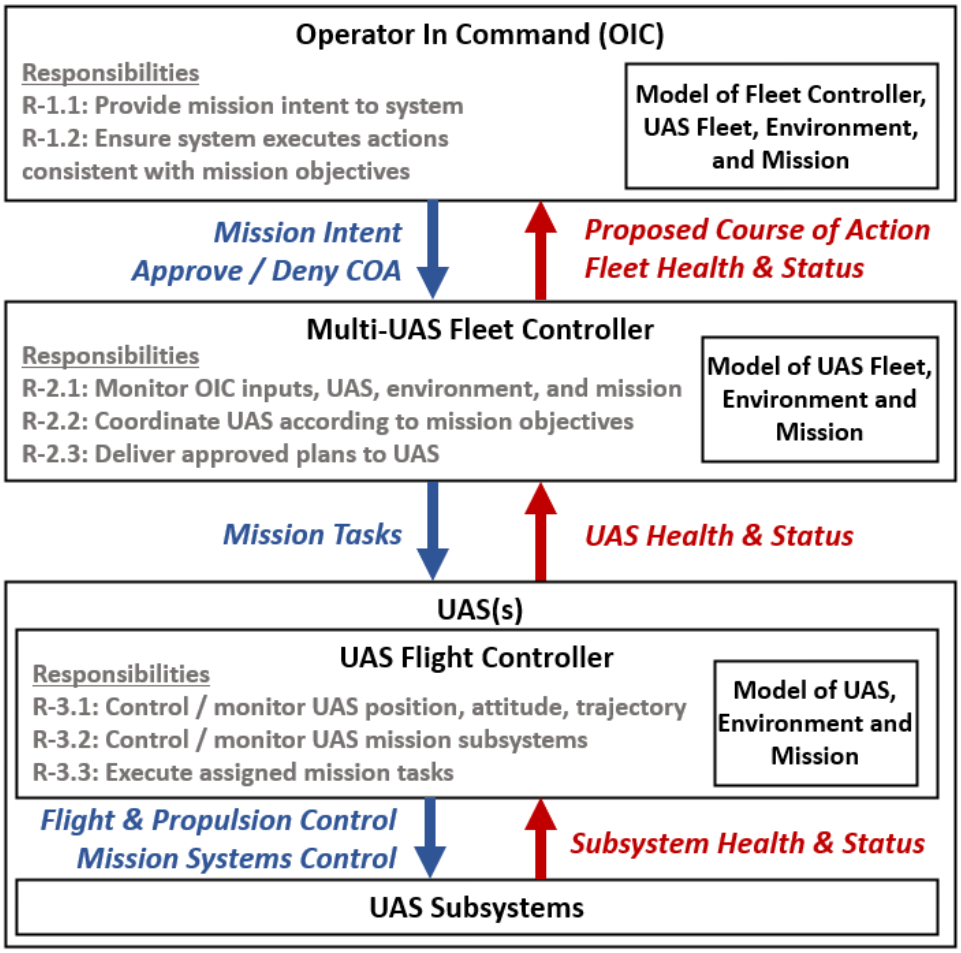


Figure 2-8: STPA Example Multi-UAS System Control Structure (adapted from [143])

Step 3 identifies unsafe control actions (UCAs), which are control actions that, in a particular context, and worse-case environment, will lead to a hazardous state [50]. There are four possible ways to consider how each control action in the control structure can lead to a hazard: (1) not providing the control action, (2) providing the control action, (3) providing a safe control action but too early, too late, or in the wrong order, and (4) providing a control action for too long or stopping it too soon. Table 2-3 shows one example of each type of UCA for the multi-UAS system example [143]. Many UCAs of each type can be derived.

Table 2-3: STPA Example Multi-UAS Unsafe Control Actions (UCAs) (adapted from [143])

UCA Type	UCA	Hazard Traceability
Not Providing	[UCA-1] Operator *does not provide* "Approve COA" when the COA fulfills mission objectives	H-1
Providing	[UCA-2] Operator *provides* "Approve COA" when the COA has UAS violate minimum separation	H-2
Too Early / Late / Wrong Order	[UCA-3] Operator *provides* "Approve COA" *too late* when the COA no longer fulfills mission objectives	H-1, H-2, H-3
Applied too long / Stopped too short	*Not applicable for this analysis because "Approve COA" is a discrete command*	*N/A*

Finally, **Step 4** identifies causal scenarios (CS) that can lead to unsafe control actions and hazardous states. This is accomplished by systematically exploring potential breakdowns in the feedback control loop in the model. The following example scenario is derived from UCA-3 in Table 2-3. Many scenarios may be found for each UCA, and some scenarios may be linked to multiple UCAs.

> CS-1: The Fleet Controller updates the COA request so frequently that the operator cannot assess its validity before it is replanned. Thus, the operator is in a cycle of perpetual COA review. [UCA-3] [143]

The output of the analysis is a set of constraints to eliminate or mitigate UCAs and causal scenarios. These can form the basis for requirements or recommendations to build safety into the design throughout its lifecycle. For example, the following design requirement (DR) can be derived from CS-1.

> DR-1: The system must not enter a state where the operator cannot provide input because of a perpetual COA update. [UCA-3, CS-1] [143]

2.4.2 Why Use STAMP for this Research Topic

There are several reasons why the system-theoretic foundation of STAMP and STPA are well-suited to address some of the challenges in fielding novel aerospace systems with more complex component interactions and dynamics. The following strengths enable the holistic modeling and analysis necessary to design and assure such systems are safe.

First, the STAMP causality model can handle non-linear or circular relationships between system components, unlike the chain-of-event models described in Section 2.3.1. This is critical

to model collaborative relationships between system controllers engaged in joint activity. As previously described in the Coactive system model (Figure 2-4), teammates reciprocally help close each other's control loops [2]. Linear causality models simply cannot represent this fundamental relationship. While Systems Theory does underlie other analysis approaches, such as System Dynamics [185] and Cynefin [186], these methods were developed to model social systems, whereas STAMP is more appropriate for engineered systems.

The second attribute that makes STAMP appropriate for analyzing the safety of collaborative systems is its ability to explore interactions between hardware, software, and human controllers. Traditional hazard analysis tools are more hardware-focused and require separate approaches for software and human factors. However, the separation of analysis for these different types of components is reductionist and, therefore, incapable of properly predicting the emergent property of system-level safety. STAMP provides the necessary unified approach to integrate the different technical domains that influence these collaborative interactions.

The third advantage of STAMP is that it does not assume that accidents are only the result of random component failures like most other techniques do. It also considers how many accidents have occurred as a result of unsafe relationships between fully functioning entities [38]. This is important because software does not fail. It always executes exactly as it was programmed, and unsafe behaviors that arise from it can almost always be traced to flawed requirements [144]. In addition, some system failures are not hazardous and do not lead to accidents. From this perspective, STAMP remains focused on system safety rather than on factors that may only create a reliability concern [38].

The fourth reason is that STAMP emphasizes system context. All engineered systems are integrated within a larger socio-technical system which must holistically exhibit safe behavior. The non-traditional human and machine roles proposed in many novel aerospace systems need to be analyzed in the context of the broader aviation system they will operate in. It is insufficient only to analyze the interactions between team members, which is what most of the teaming models and design processes described in Sections 2.1 and 2.2 focus on.

The fifth strength of STAMP and STPA is the ability to use abstraction to model and analyze arbitrarily complex systems. This addresses the scalability concerns found in leading collaborative system design approaches like Dearden [46] and Coactive Design [2]. It is also conducive to first analyzing what the system or team must accomplish as a whole, as recommended in several design techniques, and then using refinement to explore functional allocation within its components.

Finally, abstraction enables STPA to be applied at the very early stages of design, when little design detail is available, and where safety requirements can be the most effective. This can facilitate early traceability of architecture requirements and support V&V to integrate safety assurance into the design process. Such an attribute is necessary to enable an assurance by construction paradigm that may overcome some of the limitations associated with certifying complex systems today.

2.4.3 Previous Relevant STAMP Work

The System-Theoretic foundation of STAMP provides a general framework that supports modeling and analysis of any type of system and any type of emergent property. However, extensions of STAMP and STPA have been developed to provide additional guidance when needed for specific problems. Examples include techniques to focus on system security [187], adding depth to analyzing human factors [188], [189], and providing a framework to use STPA actively during operations [190].

In this light, the following explores past STAMP work related to systems that exhibit collaborative behaviors. The goal is to provide orientation on where additional guidance is needed to address this topic systematically and rigorously.

STAMP Extension for Coordination

STPA-Coordination was developed by Johnson [191] to enhance the identification of causal factors when multiple controllers engage in coordinated activity. The framework identifies four fundamental coordination relationships, shown in Figure 2-9, and nine elements necessary for coordination, listed in Table 2-4. Occurrences of missing, inadequate, or late establishment of these elements help identify flawed coordination cases that could lead to system hazards.

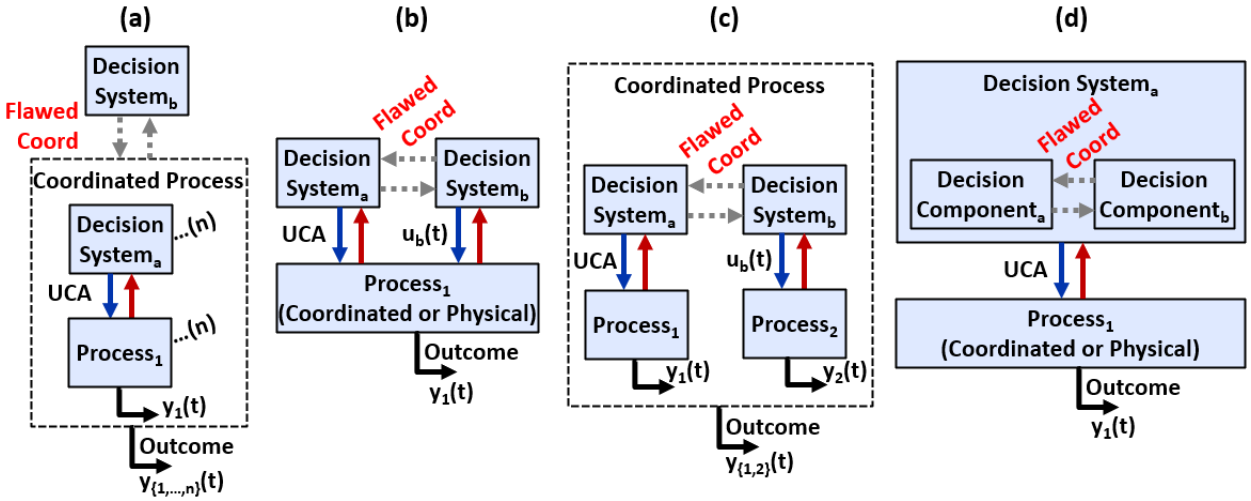


Figure 2-9. Fundamental Coordination Relationships (adapted from [191])

Table 2-4. Nine Coordination Elements Defined by Johnson [191]

Coordination Components:	Goals Strategy (Activities) Decision Systems
Enabling Processes:	Communications Group Decision Making Observation of Common Objects
Enabling Conditions:	Authority, Responsibility, Accountability Common Understanding Predictability

Johnson's framework is strongly related to this research topic because coordination is one of the attributes required for effective team interactions [59]. For this reason, several of the coordination elements are accounted for in the techniques introduced in this work.

However, the scope is different as not all coordinated systems are necessarily collaborative. A common type of collaborative relationship sought in novel systems is represented by Case (b) in Figure 2-9. Given recursion in Systems Theory [182], Cases (c) and (d) can arguably be treated as abstractions of (b) from the perspective of collaborative relationships. Conversely, Case (a) represents the centralized or supervisory control interactions that are widely fielded in traditional systems and therefore, not the focus of this research.

Additional work is needed to rigorously consider fundamental dynamics exhibited when Case (b) behaves as a collaborative system with interactions akin to those in human teams. For example, clear guidance must be provided on handling shared control, alignment of team cognition, and changes of authority. Similarly, Case (b) may involve a hierarchy between the controllers, which may even be dynamic, as considered in Mixed-Initiative Interactions [67].

Johnson also recognizes that more specific direction is required to consider coordinated interactions between humans and machines [191]. A clear path to integrate this model into a broader socio-technical system and guidance to refine the analysis to different levels of interest systematically is also necessary. Finally, a framework is needed to use the outputs of the analysis to guide architectural design and assurance processes.

Observations of Relevant STPA Analyses

In addition to Johnson's work, multiple recent analyses have applied STPA on systems that exhibit some of the complex interactions of interest in this research. Eight of these were reviewed to study their types of causal relationships and examine how they were addressed in the analysis.

Robertson [53], Montes [188], and Horney [192] analyzed interactions between human-piloted aircraft and UAS executing military missions and formations using STPA. Abrecht [193] and Mackovjak [194] studied the design and accidents of similar concepts in the naval domain. Kharsansky explored architecture options of multi-satellite constellation systems [18]. Peper applied STPA to an assembly system with three remotely operated vehicles that collaboratively transport a large product [195]. Finally, Wong used STAMP to investigate communication breakdowns that occurred when two airliners collided after receiving conflicting directions from Air Traffic Control (ATC) and the Traffic Collision Avoidance Systems (TCAS) [196].

Because these studies were all performed for different purposes, comparing them to one another or to the intent of this research is unfair. Furthermore, not all of these systems were fully representative of the novel aerospace systems introduced in Chapter 1. However, several common trends and challenges related to collaborative control were noted.

In most studies, the lowest level of the hierarchal control structure stops at the collaborative entities, such as the multiple UAS or satellites, and does not show the shared mission or formation they are collectively responsible for controlling. Many of the studies shy away from analyzing the distributed control that occurs when each individual controller makes its own decisions about how it controls the shared process using information received from its peers.

The definition of Unsafe Control Actions (UCAs) in STPA tends to focus the analysis on one controller and one controlled process at a time, whereas teaming involves a collection of control actions from multiple controllers. Finally, mechanisms associated with changing control responsibilities and shared process models between multiple controllers are not explicitly considered. Such challenges present an opportunity to extend how STAMP models are developed and analyzed to more systematically address interactions exhibited in collaborative control.

2.5 Summary of the Literature

This Section reviewed a wide basis of literature related to engineering systems that exhibit collaborative interactions inspired by human teams. The key takeaways are the following.

First, there are many ways to characterize and model teaming interactions. Some properties are common across different team compositions, whereas some are more specific to human, human-machine, or multi-machine teams. However, most models that describe some of the intricacies of teaming dynamics lack actional guidance for system designers and do not focus on safety.

Second, there are very few systematic processes available to guide the architectural design of interdependent systems. The majority of the frameworks are based on high-level or reductionist design patterns which offer little rigor in reasoning about the system as a whole. The most advanced methods provide some mechanisms to reason about systems holistically, but they face various drawbacks in being able to design safety top-down.

Third, conventional safety assurance processes are generally expensive, are applied too late in the development life cycle and are inadequate to handle interdependent and collaborative systems. This means there is currently no effective path to certify many of the novel systems introduced in Chapter 1.

Finally, Systems Theory and STAMP provide an avenue to integrate system design and safety assurance activities from the earliest phases of development. Their principles were shown to be well-suited to handle the collaborative and interdependent relationships of interest in this research. However, they require additional guidance to rigorously and systematically handle the complex attributes associated with these interactions. The next two chapters extend their analytical scope to address this need.

Chapter 3: Defining Collaborative Control using Systems Theory

Despite the many strengths STAMP-based techniques possess to analyze the safety of complex systems, some of the types of complex interactions sought in novel aerospace designs have not been specifically defined using Systems Theory. The lack of definition in the underlying causality model means the hazard analysis may omit certain causal factors or not handle others systematically. This chapter defines the collaborative control interactions observed in teaming so that they can be more completely analyzed using STAMP and the analysis tools built on it.

A widely cited *team* definition is: "A team consists of two or more entities who interact dynamically, interdependently, and adaptively toward a common and valued goal, with unique roles and functions to perform" [51, p. 3]. It is nearly identical to that of a *system* (see Chapter 1) and, therefore, is not very useful by itself in distinguishing teams from other systems. Similarly, characterizing teamwork as "more than the aggregate of individual behaviors," [58] defines it as an emergent property, which is again fundamental to systems. This means teams can be defined as identical to systems.

So why are the new human-machine teaming or multi-machine teaming concepts proposed in aerospace so challenging to engineer? What makes them different from other systems that have been successfully fielded? The reason is that their component interactions are more complex and/or less well understood than those in previously fielded systems in safety-critical applications.

Figure 3-1 conceptualizes some of the differences in the types of interactions exhibited by human teams compared to those in existing human-machine systems in a similar setting. The left is a traditional Human-Automation Interaction (HAI), with a pilot supervising an autopilot to control an aircraft. The pilot delegates some control authority to the autopilot using predefined modes and sets goal conditions accordingly. Beyond that, the control structure is static, and the autopilot-aircraft feedback control loop is not closed by the pilot.

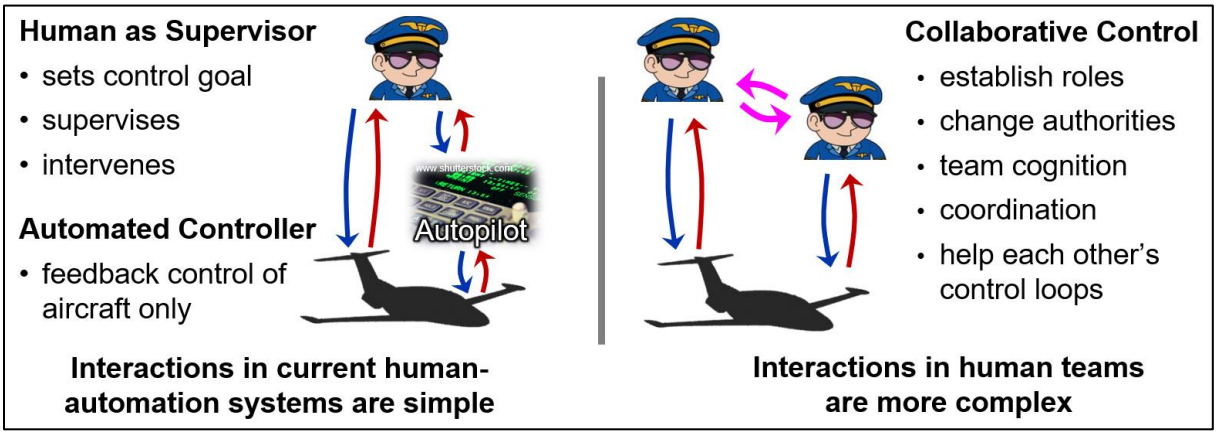


Figure 3-1: Contrasting Interactions in Fielded Human-Machine Systems and Human Teams

Conversely, the right shows a human flight crew controlling the aircraft. Some interactions are supervisory in nature, such as the captain delegating flight control roles. However, others are more collaborative. The pilots are part of each other's control loops in operating the aircraft as they coordinate, direct each other, transfer tasks, and mutually monitor their activities. This collaborative control paradigm observed in human teaming goes beyond what current fielded human-automation systems are able to exhibit. But these teaming interactions are inspiring novel system concepts.

To analyze the safety of such novel systems, it is necessary to more precisely identify the complex human interactions described in the example. A few taxonomies have previously been proposed to describe system and component interactions for the purpose of hazard analysis. For example, Perrow [197] classifies system interactions using two dimensions: (1) tight vs loose coupling and (2) linear vs complex. However, this framework is highly subjective, and the theory it supports emphasizes that accidents occur due to component failures, which is inconsistent with modern system safety.

More recently Saurin & Patriarca [198] developed a 9-dimensional taxonomy for socio-technical system interactions. The axes include: nature of agents, output nature, levelling, waiting time, distance, degree of coupling, visibility, hazards, and parallel replication. This model is specifically designed to feed Functional Resonance Analysis Method (FRAM). Unfortunately, this technique assumes sequential, or acyclic, causality, which is an oversimplification of non-linear interactions that contribute to accidents [4].

Other research domains have categorized system interactions, but not necessarily for safety analysis. HAIs are often described along the popular Levels of Automation (LOA) axis [122], which is helpful to conceptualize how automation is intended to support humans in a design. However, as reviewed in Chapter 2.2, the LOA construct is too simplistic and does not characterize collaborative behavior [120]. Distributed control theory is employed by Murphey & Pardalos [79] to define increasingly coupled interactions between multiple controllers, ranging from collective to coordinated. Finally, multi-agent control theory is used by Parunak et al. [199] to classify interactions according to the different types of communication, intent, and congruence of system-level goals.

Various elements of these frameworks helped inspire the formulation proposed in this dissertation. However, these sources are still insufficient to describe the teaming interactions reviewed in Chapter 2.1 using Systems Theory. This requires a new taxonomy to be developed.

The system-theoretic framework introduced in this chapter consists of (1) a taxonomy of the structure of interactions between multiple controllers and (2) a set of dynamics observed in collaborative control. It creates the necessary foundation to extend system-theoretic hazard analysis methods to systematically identify causal factors associated with these interactions.

3.1 Taxonomy of the Structure of Controller Interactions

The dynamics between multiple interacting system entities, including those observed in collaborative control, are influenced by the structure of the interaction. A taxonomy of seven

structural dimensions was developed to help reason about causal relationships between controllers (Figure 3-2). The taxonomy is inspired by the literature covered in Chapter 2, a review of various aerospace systems, and discussions with researchers that provided various perspectives and examples on teaming systems[4].

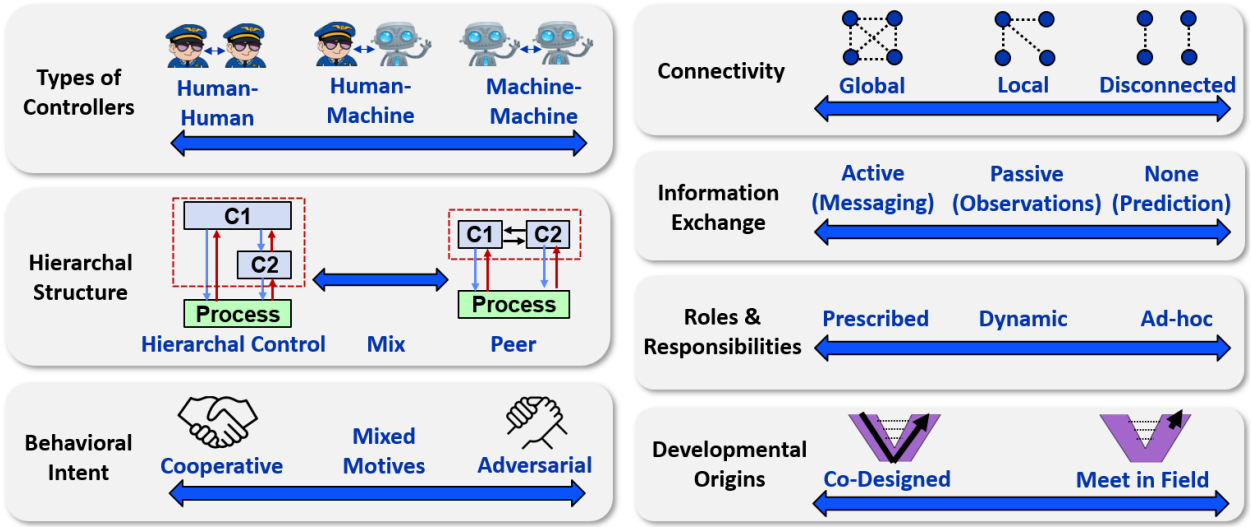


Figure 3-2. Taxonomy of Structure of Interactions between Multiple Controllers

First, the dynamics are shaped by the *types of controllers* [A] interacting, which may be humans only, human(s) and machine(s), or machines only. These interactions can also form the basis of higher-level systems, such as human organizations, teams of teams, or multiple human-machine systems interacting as a whole. Finally, the types of controllers may not be determined yet if the system is in design. The nature and differences between interacting controllers have significant implications for how they coordinate and make control decisions. It is noted that the framework could also include interactions with other biological controllers, such as pets or viruses, but those are omitted because they are less relevant to aerospace systems.

Next, the *hierarchal structure* [B] between controllers can vary from a *hierarchal control* relationship to a partnership between *peers*. Some interactions can exhibit a mix of both traits. For example, interactions among human pilots involve both hierarchal control and peer partnership at different levels of work. The captain directs the roles and responsibilities of the first officer (FO) by designating her/him as Pilot Flying versus Pilot Not Flying. However, the captain and FO also work together at a lower level to aviate, navigate, and communicate.

Entities can exhibit a range of *behavioral intent* [C] toward one another. Designed systems often consist of components that are mutually cooperative to achieve a common objective. On the opposite end, some systems contend with adversarial interactions, such as those found in competition and conflict, as described by Parunak [199].

[4] Discussions with Emily Cowen, Andrew Heier, Kyle Ingols, Reed Jensen, Dr. Kevin Lahey, Dr. Caroline Lamb, Dr. Vincent Mancuso, Col (ret) Mike Pietrucha, and Dr. Rohan Paleja helped, in part, shape the taxonomy. Thank you.

Between these two ends, there are also interactions without reciprocal intent. For example, two entities may consist of one behaving cooperatively whereas the other is not, as is the case in formation rendezvous with a non-cooperative target. Adversarial intent can also be unilateral, as observed in insider threat scenarios. Furthermore, interactions may also be influenced by mixed motives, as is experienced between drivers in merging traffic where a lane ends. Drivers must cooperate to allow each other through and avoid crashing into each other, but they are also driven by non-cooperative intentions to reach their respective destinations on time.

Multi-controller interactions are also influenced by their level of *connectivity* [D]. Connectivity may be *global,* in which all controllers involved in joint activity can directly exchange information with one another. It can also be *local*, where two controllers that are not directly connected may exchange information indirectly through other entities. Finally, some controllers or groups of controllers may be disconnected and not be able to exchange information.

Next, there are different types of *information exchange* [E] that can occur along these connections, as described by Murphey [79]. Communications can involve active messaging where content, such as controller states and intentions, is encoded and shared. Communication can also be passive, in which a controller observes the behavior of another to estimate its state [79]. Finally, the existence of lines of communication does not mean information exchange actually takes place. The activity of multiple controllers can be coordinated a priori in an open loop via preplanned policies and behavioral predictions.

The roles and responsibilities [F] of interacting entities can range from prescribed to ad-hoc. In simpler systems, the responsibility, authority, and accountability for controllers are prescribed using modes so that control boundaries are defined to avoid overlaps. For example, a pilot can turn on "heading hold" mode on an autopilot to entirely delegate that task to the machine.

Other systems may intentionally allow more dynamic control boundaries with overlap between multiple entities to improve performance. For example, there are many concepts of teams of multiple robots that share a common set of tasks and must coordinate and deconflict task allocation during operation. Furthermore, roles and responsibilities may even be ad-hoc and determined only during execution. This ad hoc teaming occurs in human interactions, for instance, when a team forms to handle a roadside accident.

Finally, interactions are also influenced by their *developmental origins* [G], i.e., the time when they are considered in the engineering lifecycle. Some controllers are co-designed to interact from the beginning, while for others, new functional interactions are only considered post-fielding. In between, this axis also includes separate designs of interacting controllers according to a common specification. It also contains unilateral relationships, where one controller is developed to interact with another that is already fielded.

Categorizing the interactions between multiple controllers according to these structural dimensions does mean assigning a value on a numerical scale. The *Types of Controllers* axis represents nominal data, and the remaining six axes are ordinal. Some interactions may exhibit multiple labels on each axis. Furthermore, there is no claim that these axes are independent. The purpose of this taxonomy is to qualitatively consider structural factors that influence causality between multiple interacting controllers.

3.2 Collaborative Control Dynamics

To determine the causal effects of different types of interactions in hazard analysis, those relationships must be included in the underlying causality model. The system-theoretic foundation of STAMP provides a mechanism to model causality in any arbitrary control system, no matter how complex. However, as highlighted in Chapter 2, STAMP-based techniques like STPA need additional guidance on how to systematically handle some of the more complex team-inspired interactions.

To address this gap, this section defines nine dynamics that are observed in collaborative control systems. These interactions are derived from the teaming fundamentals reviewed in Chapter 2 and the study of novel aerospace systems described in the literature. The definitions are grounded in the principles of Systems Theory and are formulated using STAMP. The structural dimensions in the taxonomy above influence if and how these dynamics are exhibited by a system.

Using Systems Theory, the concept of emergence implies that the collaborative control dynamics can only be described in terms of the interactions among multiple entities. In this case, the interacting components are two or more controllers engaged in joint activity. The collaborative control dynamics are irreducible in that they have no meaning for any individual controller. The concepts of hierarchy, communication, and control are all explicitly represented in the models shown in Figure 3-3.

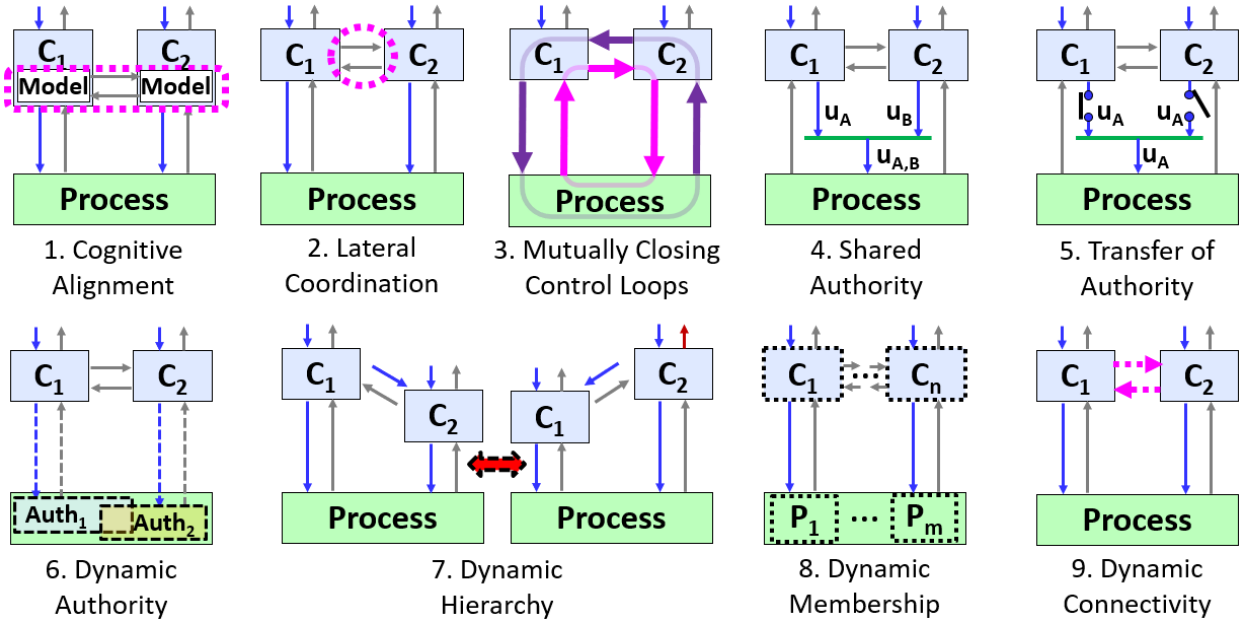


Figure 3-3. System-Theoretic Collaborative Control Dynamics

The collaborative control dynamics are defined according to the three parts of STAMP. The causal relationship between the controllers is represented using a *hierarchal control structure*. Each controller has a *process model* which contributes to the system behavior. By adequately analyzing the system, *safety constraints* can be identified to enforce the safe collective behavior of the team.

The following discussion describes what the collaborative control dynamics mean in STAMP models. Examples of each of these interactions are presented in Section 3.3.

1. Cognitive alignment is the process multiple controllers use to establish and maintain consistency in their process models and control decisions. The dynamic can involve synchronizing information held in common as performed in shared situational awareness [60]. It may also require controllers to leverage distributed situational awareness by coordinating about who knows what and when within the team [61].

As emphasized in STAMP, accidents often occur because controllers have flawed models of the process they are controlling [38]. For collaborative control, this concept is extended to account for how the models of multiple controllers sharing a process may be flawed relative to one another. Any single controller may not have a full model of the controlled process and may depend on model variables held by its collaborators. Furthermore, a controller may consider estimates of the states of the collaborating controllers as part of their decision-making.

Mutual confidence is also part of cognitive alignment. It relates to mutual trust, a key mechanism for effective teamwork [59]. Confidence is relevant in this research because it is encodable in machines, unlike trust, which is inherently a human property. As proposed for trust by Chancey et al. [78], confidence is an element of the process model for each controller regarding the expected behavior of other interacting controller(s). Levels of confidence can be mutually assessed for the behaviors of interacting entities. A controller's confidence in other controllers may influence its own control decisions.

The ability of controllers to align their cognition is heavily influenced by different dimensions in the taxonomy introduced in the previous section. The process is more challenging if the controllers are of different types. The hierarchy, connectivity, and information exchange define the mechanisms available to maintain the alignment. The rigidity of controller roles and responsibilities may affect the complexity of the process models that need to be synchronized. Finally, the developmental origins can impact the compatibility of the cognitive functions in multiple controllers.

2. Lateral coordination is a causal process between peers that does not imply control [191], [199]. It describes how a component provides information to another component in a way that influences its behavior, but without imposing constraints. Controllers can achieve this dynamic using communication exchanges that are deliberately intended to facilitate coordination. For example, pilots of multiple aircraft at an uncontrolled airport laterally coordinate by radioing their position and intentions.

Lateral coordination can also involve passive observations. Controllers that monitor each other's behaviors can be implicitly influenced by the interpretation of information that was not deliberately provided for coordination [200]. In the same example as above, pilots at an uncontrolled airport also observe each other visually and consider implicit factors (e.g., their tone of voice) to update their models of each other.

Lateral coordination is one of the mechanisms that may be used by a team to maintain cognitive alignment. It is distinct from vertical coordination, as defined by Johnson [191], which occurs when a central controller coordinates the decisions of the components over which it has authority. An example of this is when Air Traffic Control coordinates the trajectories of multiple aircraft with respect to each other.

The lateral coordination dynamic is enabled by the hierarchal structure of the interacting controllers, as described in the taxonomy. It is also supported by the connectivity and the ability to actively communicate. Johnson's extension of STPA proposed a framework of nine coordination elements to consider as missing or inadequate when analyzing safety in coordinated systems [191]. The current work employs some of these ideas to consider how coordination can lead to unsafe collaborative control.

3. Mutually closing control loops is a central concept in interdependence analysis in coactive design [2]. Controllers observe, predict, and direct each other bi-directionally to execute joint activity (see Figure 2-4). In a STAMP control structure, this dynamic indicates a controller may depend on another controller to receive feedback about the process it is controlling. It also means a controller may depend on feedback from its controlled process to determine how to interact with a collaborator. The interaction implies that controllers are inherently part of each other's feedback control loops, and as such the actions of those controllers cannot be analyzed individually.

4. Shared authority occurs when multiple controllers have authority over one process or multiple interdependent subprocesses. This dynamic is common in hierarchal systems, in which a supervisor (typically human) can delegate certain functions to another controller under its supervision (often automated systems) but can also intervene and reclaim control. However, shared authority can also occur when controllers interact as peers. In some cases, a system may require simultaneous inputs from multiple controllers to control the same process.

In STAMP, shared authority is fundamental to modeling collaborative control as it describes the joint activity executed by multiple controllers. The key implication of this dynamic is that hazards may occur because of how multiple control actions are provided collectively to the shared process. Therefore, these control actions must be analyzed in the context of one another.

5. Transfer of authority is a dynamic that can only occur in the presence of *shared authority*. It specifically describes how some systems perform handoffs of a common control action between multiple controllers over time. Handoffs may be done to dynamically reallocate control authorities to address shortfalls identified during operation. Not all systems with *shared authority* exhibit *transfer of authority.*

6. Dynamic authority also requires the presence of but is not implied by *shared authority*. Dynamic Authority enables multiple controllers with overlapping control boundaries to shift the division of labor during the execution of joint activity. As previously described, some systems are intentionally designed with this overlap so that they can adjust the allocation of tasks dynamically and improve performance.

The ability to adapt, redirect resources, and change collective behavior are important in effective teamwork [59]. However, dynamic authority can lead to conflicts or gaps in authority, which are well-known causal factors that lead to hazardous behavior in collaborative control [38], [201]. The significance of *transfer of authority* and *dynamic authority* in STAMP is that the authority of some of the controllers to provide some of their control actions will vary over time.

7. Dynamic hierarchy occurs in mixed-initiative interactions [67]. Controllers alternate in leading the team to execute various parts of the joint activity. This dynamic can be beneficial in improving performance when certain controllers are contextually better suited to control others. Such interactions are commonly observed in human dialog and are of interest for designing

collaborative systems [67]. The implication of dynamic hierarchy in STAMP is that some systems have multiple controllers that have the ability to provide control actions to one another.

8. Dynamic membership means the set of controllers engaged in joint activity can change over time. Collaborative teams may lose, replace, or gain members during activity. This dynamic may require controllers to track the set of active collaborators as a process model variable. Similarly, the controlled process may be composed of dynamic subprocesses that come and go over time. As a consequence of dynamic membership, STAMP models must account for the variable participation of some of its controllers.

9. Dynamic connectivity indicates the network topology among controllers changes over time while the controllers are interacting. Most systems may be subject to failed channels of control, feedback, and information flow, as currently emphasized in STAMP. However, this dynamic specifically focuses on systems for which these channels are expected to vary as part of the operation. It implies that both global connectivity and communication links between any controller pair are not always guaranteed. Note that dynamic connectivity can also apply to the controlled subprocesses and will affect dynamic membership.

In addition to these dynamics, the collaborating controllers $\{C_1,...,C_n\}$ have their own accountabilities, which may be different from one another. This concept is important because the system is open and interacts across its boundary with a larger system, which also has causal implications. Figure 3-3 accounts for this notion by including separate feedback control connections to higher authority controllers for each of the controllers shown.

3.3 Analyzing Systems for Collaborative Control Dynamics

The system-theoretic framework introduced in this chapter was evaluated on a set of 101 component interactions that are part of aerospace systems. These systems, which are described in the literature and listed in Appendix 1, represent both fielded systems and unfielded systems, e.g., systems in concept development, systems that have been prototyped but not yet fielded, etc. Most of the systems were encountered while reviewing the teaming literature, and they include all of the motivating examples listed in Chapter 1. However, the set is not necessarily representative of all possible and actual aerospace systems.

The analysis explores interactions between the main entities for each of the sampled systems. Some systems have multiple interactions to categorize. For example, the Aircraft Collision Avoidance System (ACAS-X) involves interactions between aircraft, aircraft flight crews, and Air Traffic Controllers (ATC), where components can also interact with one another. Each of these interactions can be categorized differently (see Example 1 below).

The goal of this analysis is (1) to demonstrate how to categorize controller interactions, (2) to compare the presence of collaborative control dynamics in different types of systems, and (3) to describe the relationships observed between dynamics.

3.3.1 Demonstration of the Framework

Two examples illustrate how to categorize the structure of interactions and the presence of collaborative control dynamics in multi-controller systems. This section also describes how to handle some of the nuances encountered in the set of interactions analyzed.

Example 1: ACAS-X Aircraft Interactions

The first example is the interaction between aircraft using the Airborne Collision Avoidance System (ACAS-X). ACAS-X is a series of developmental upgrades to the fielded Traffic Collision Avoidance System (TCAS) to handle a wider range of aircraft operations [138], [202]. ACAS-X units on different aircraft interact as a collaborative machine team to jointly ensure collision avoidance. The categorization shown in Figure 3-4 and described below focuses on interactions between ACAS-X controllers on deconflicting aircraft. It does not describe ACAS-X to Flight Crew or ACAS-X to ATC interactions, which would be categorized differently.

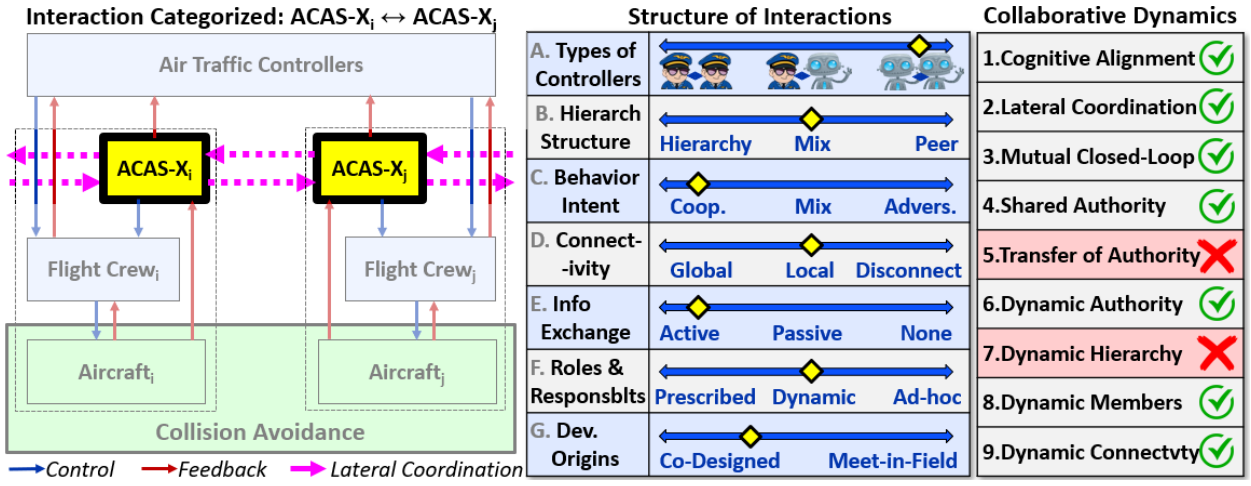


Figure 3-4. Categorization of Interactions between ACAS-X Aircraft

The structure of interactions is categorized according to dimensions [A-G] as follows. First, the types of controllers [A] are machines, which consist of transponder-computer units that autonomously exchange information with one another. These machines communicate their decisions to human flight crews and ATC using other interactions.

The hierarchal structure [B] between these controllers exhibits both hierarchal control and peer-level interactions. A hierarchy is established between aircraft based on their transponder identifiers to allow the higher-ranked aircraft to overrule the lower-ranked aircraft in certain situations. However, the lower-ranked aircraft may also share its resolution intent first, in such a way that influences the higher-ranked aircraft's decision.

The behavioral intent [C] between ACAS-X controllers is cooperative. The connectivity [D] in networks of ACAS-X aircraft is local as resolution advisories are coordinated between aircraft pairs only. The information exchanged [E] consists of active coordination with content-based messages.

The roles and responsibilities [F] of the controllers are dynamic. In every encounter, the entities must determine their relative hierarchy, which specifies their respective responsibilities and authorities. Finally, the developmental origins [G] of individual ACAS-X systems are not all co-developed, but they are all designed according to a common standard intended to work collaboratively.

Aircraft interactions using ACAS-X exhibit seven of the nine collaborative control dynamics. They facilitate cognitive alignment through shared information about the state and intended behavior of each entity. This alignment also includes mutual confidence because logic is in place to assess whether a peer is executing the expected deconfliction maneuver. The cognitive alignment employs lateral coordination in the form of exchanged resolution advisories and position information. They mutually close each other's control loop of collision avoidance by selecting both aircraft control inputs and messages to one another based both on their relative position and their exchanged intent.

This multi-controller system exhibits shared authority over the controlled process of collision avoidance. Dynamic authority is also observed as each ACAS-X controller pair must determine its hierarchal structure, and as a result, define during operation the control boundaries of each participant. The team also has dynamic membership as the set of aircraft that needs to mutually deconflict changes throughout execution. Finally, there is dynamic connectivity given that ACAS-X aircraft cannot assume fully reliable communications.

Some collaborative control dynamics are not observed between interacting ACAS-X aircraft. First, ACAS-X controllers do not transfer the authority of their controlled process over to other ACAS-X controllers. When interacting, each ACAS-X controller maintains its control authority over collision avoidance, which is actuated by its flight crew controlling the aircraft according to its resolution advisories. Similarly, the system does not exhibit dynamic hierarchy. While two ACAS-X units interact, there is no provision for one to be the ranking unit for part of the time, and then dynamically switch this role with the other.

Example 2: Human-Digital Copilot Interactions

The second example categorizes the interaction between a human pilot and an automated *digital copilot* featured in a variety of concepts that aim to simplify aircraft piloting operations (Figure 3-5). Specifically, the concept proposed by Dropkin et al. [22] involves a human and an automated assistant that collaboratively execute checklists. The human can delegate checklist execution to the assistant, as s/he would with a human copilot. From there, the automation can request the human to perform certain checklist actions while it takes care of others.

With regards to structural dimensions [A-G], in [A], the interaction is between a human and a machine. For [B], this system features a mix of hierarchal control and peer interactions. The human can delegate tasks to the automation in a supervisory relationship. However, within those tasks, the automation can request assistance, ask the human to execute certain subtasks, seek and provide clarifications, and monitor human execution as a peer. Next, the interaction is mutually cooperative in intent [C].

For [D], connectivity is global. While this case only involves two controllers, and therefore global connectivity is arguably no different than local, the important point is that nominally all collaborators are mutually connected. Dimension [E] involves both active and passive

information exchange. Controllers actively exchange content-based messages, for example, using voice commands, and also passively observe each other's process control activities.

In [F], the system has dynamic roles and responsibilities. There is an intended overlap between the human and automation control boundaries to allow task allocation to be specified during operation. Finally, for [G], it is assumed the system interaction is co-designed. The automation is designed from the beginning to interact with a human pilot, and that the human pilot is trained specifically to work with this automation.

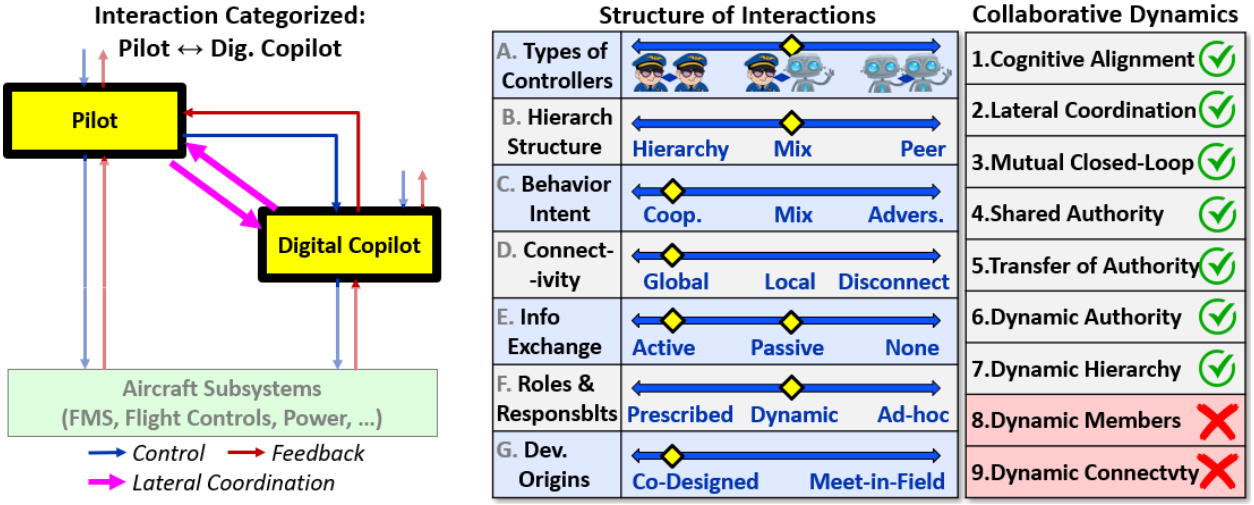


Figure 3-5. Categorization of Human-Digital Copilot Interaction

This interaction exhibits seven of the nine collaborative control dynamics. The controllers perform cognitive alignment by sharing information and relying on different knowledge they each possess about the state of the aircraft under control. There are also mechanisms for mutual assessment of confidence in peer actions. The pilot and digital copilot laterally coordinate using both deliberate voice communications and mutual monitoring of their behaviors.

As in human-crewed flight operations, they mutually close each other's control loops by receiving feedback pertaining to their control actions through their peer. Similarly, their decisions to interact with each other are influenced by feedback they receive from the aircraft. Shared authority and dynamic authority are present because controller boundaries over the operation of the aircraft overlap and must be resolved during execution.

Unlike the previous example, this interaction also exhibits transfer of authority, as one controller can hand off the responsibility to execute a task to the other. There is also dynamic hierarchy at different levels of work. The human, as a supervisor, can delegate overall checklist execution responsibilities to the automated assistant. However, the automated assistant can then direct the human to execute certain assistive tasks within the scope of the checklist.

Another way this example differs is that it does not exhibit dynamic membership or dynamic connectivity. The system prescribes a fixed set of collaborating controllers throughout execution, the human and the digital copilot. Because the controllers are collocated, their network topology is not expected to vary during nominal execution. Unsafe control paths and feedback paths in the system are sufficiently covered by STPA, and the implications of these two dynamics would not need to be further emphasized in hazard analysis.

Nuances Encountered in Categorization

The categorization of controller interactions in the sample was based on the description of the system in the literature. In some cases, insufficient information was available to label one or more of the dimensions, and those items were not evaluated for that interaction.

In other cases, the description made it possible for different versions of the system to be categorized differently. For example, one multi-UAS system explicitly stated that UAS control could be distributed or centralized [143], and therefore with or without lateral coordination. In such instances, dynamics were rated as 0.5 on the 0-1 scale of whether or not the dynamic is present. Similarly, multiple labels were assigned for the structure of interaction in such cases.

3.3.2 Results of the Categorization in Different Types of Systems

Observations noted in categorizing analyzed interactions are grouped into four categories: (1) human-machine (HM) or machine-machine (MM) interactions in fielded systems, (2) HM or MM interactions in conceptual unfielded systems, (3) human-to-human (HH) interactions in fielded systems, and (4) HH interactions in unfielded systems.

These four groups imply a categorization in the *types of controllers* dimension from the taxonomy introduced in Figure 3-2. Figure 3-6 shows the relative distribution of how the other six dimensions were labeled for each of the four groups.

Three dimensions have distributions for which HM and MM interactions in unfielded systems are closer to those found in HH interactions (fielded and not) than those seen in fielded HM and MM interactions. In the analyzed set, the unfielded systems have more *peer-level structures*, *local connectivity*, and *dynamic roles and responsibilities* in their interactions than the currently fielded systems.

Two dimensions have similar distributions across all the groups. Nearly all of the systems studied have *cooperative intent*, and most of the systems use *active communications* to coordinate activities. This observation may indicate that differences in these structural elements of multi-controller interaction are less pronounced between systems with collaborative control and those with simpler relationships.

Finally, the developmental origin axis actually aligns fielded HM and MM interactions more closely with HH interactions. In these cases, most fielded HM and MM interaction systems were developed separately to a *common standard*. Conversely, the majority of unfielded HM and MM interactions occurred between co-designed controllers. This outcome is more than likely influenced by the very nature of whether a system is fielded or still in design. Many novel systems do not yet have common standards.

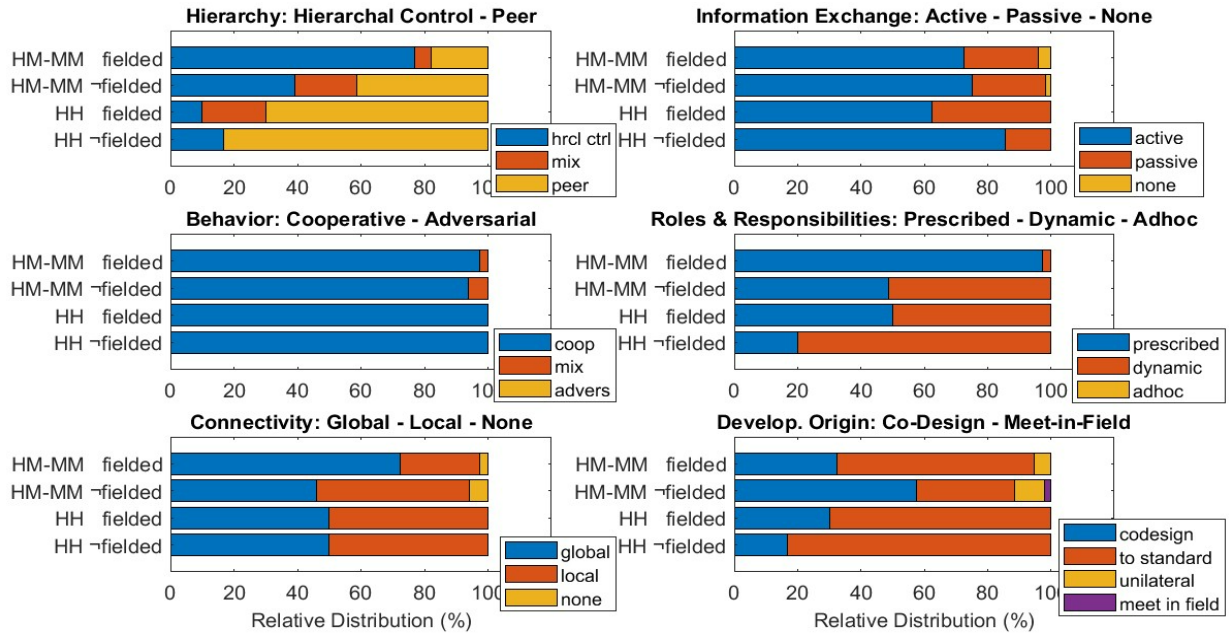


Figure 3-6. Comparison of the Structure of Controller Interactions

Next, the prevalence of the nine collaborative control dynamics is compared between the four groups. Figure 3-7 shows the mean total number of these dynamics exhibited by each interaction sampled, and Figure 3-8 captures the percentage of systems that present each dynamic. For the most part, HH interactions in the studied set exhibit more of each of these dynamics than HM and MM interactions. However, the results also indicate that HM and MM interactions in unfielded systems exhibit more of every one of the dynamics than in fielded systems in the analyzed set.

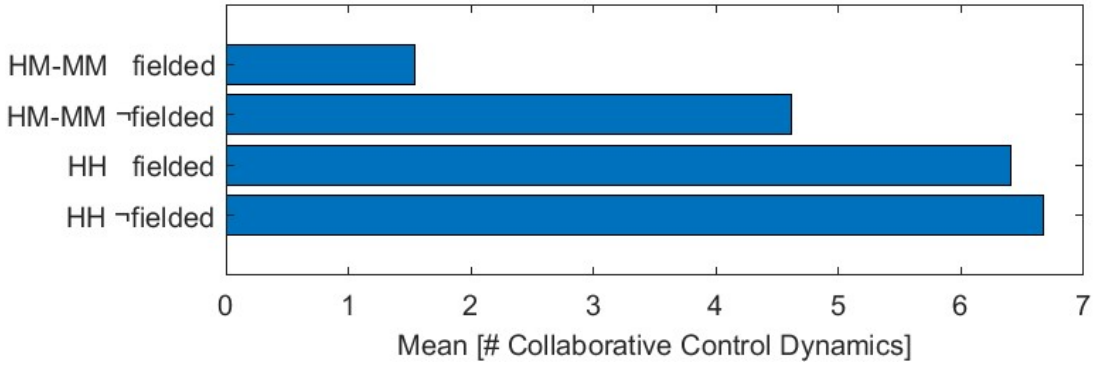


Figure 3-7. Mean Number of Collaborative Control Dynamics Found in Each Interaction

These results are not intended to quantitatively suggest that all aerospace systems follow these trends. The systems analyzed were chosen from the literature related to teaming, and they are therefore not representative of the overall population of systems.

However, there are two important takeaways from this analysis. First, there is evidence in the literature that systems are being designed to exhibit each of these complex collaborative control dynamics. And second, of the systems analyzed, those that have not yet been fielded tend

to exhibit more of these complex interactions. These points support the argument that causal factors associated with these dynamics must be considered in safety analysis and design.

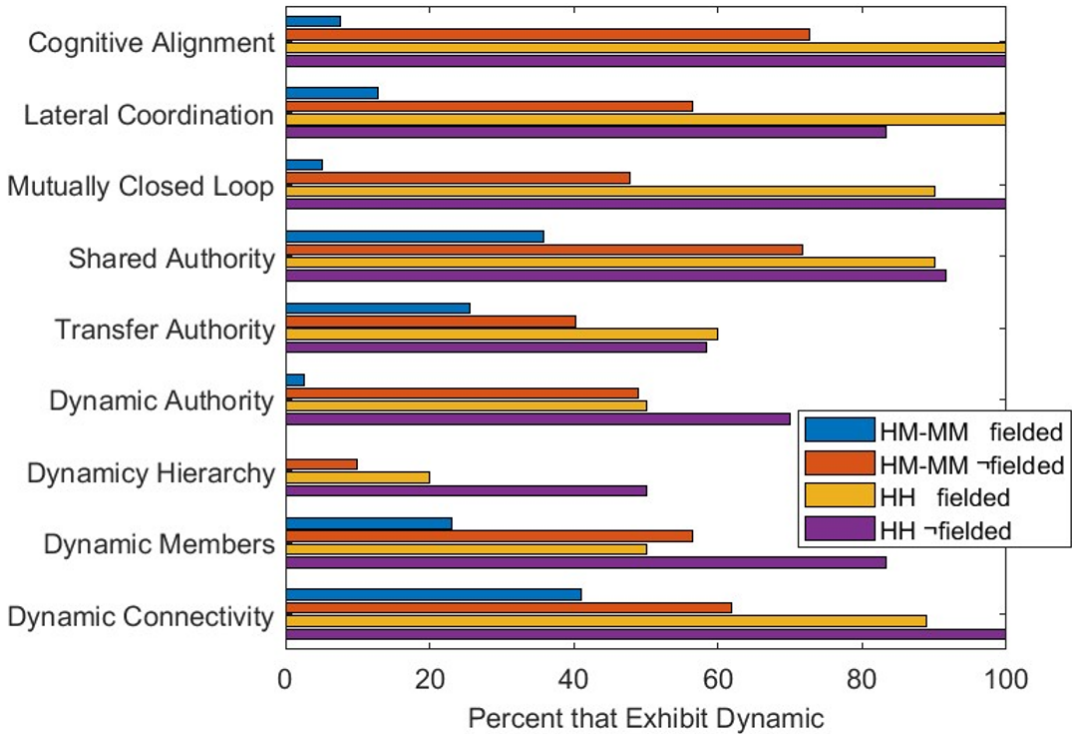


Figure 3-8. Percentage of Interactions that Exhibit Each Collaborative Control Dynamic

3.3.3 Relationships between the Categorized Dynamics

The labeling of the presence of collaborative control dynamics was compared against one another. These relationships are shown in Figure 3-9.

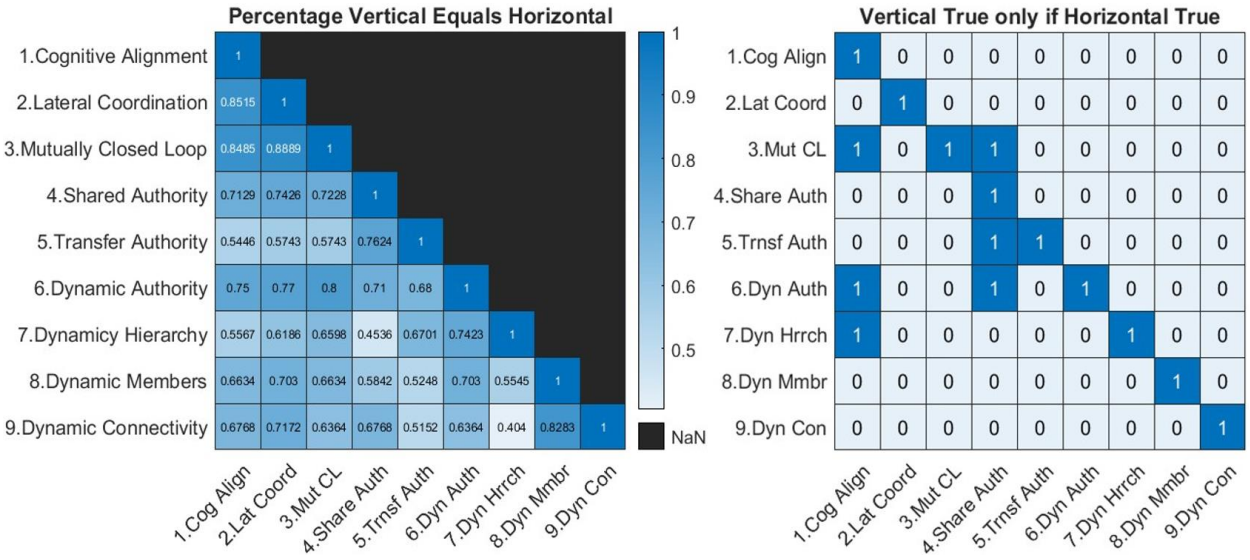


Figure 3-9. Relationship between Collaborative Control Dynamics in Sampled Systems

The left plot shows the percentage of all interactions for which two collaborative control dynamics were assessed similarly (i.e., both present or both absent). The matrix is symmetric about its diagonal, and only the bottom half is shown. The important takeaway is that no two non-identical dynamics are completely correlated in the dataset, as would be indicated by a 1 or a 0 respectively. The only 1s appear as expected on the diagonal, where each dynamic is compared to itself.

A complete overlap between two dynamics, as represented by a 1, could suggest that they are functionally equivalent when modeling the system. If so, these dynamics could be candidates to be combined. A complete lack of overlap, as indicated by a 0, could suggest that the two dynamics are mutually exclusive, perhaps due to an inverse definition of one of two equivalent properties. Neither of these instances occurred in the sample.

The plot on the right shows whether the dynamics on the vertical axis were identified only in the presence of the dynamics on the horizontal axis. Formally, the value in each cell provides the propositional logic truth value of the statement $d_{vertical} \rightarrow d_{horizontal}$ for dynamics d over the set of interactions studied. If for any of the interactions in the set $d_{horizontal}$ is false when $d_{vertical}$ is true, the cell value equals zero.

The plot shows six correlations in which one of the dynamics was only active when one of the other dynamics was also present. These are represented by the six 1s off of the diagonal. Some of these relationships were expected. For example, *transfer of authority* and *dynamic authority* imply, as defined in Section 3.2, that the system also exhibits *shared authority*.

Some of the other correlations may reflect how the dynamics are described. The plot indicates that *mutually closing control loops* only occur in the set when there is *shared authority*, which makes sense. A controller may sense feedback from the process due to an action provided by another controller on that same process. However, while this notion suggests a connection exists between multiple controllers and the same process, it does not necessarily imply that all the controllers must have authority over that process. So, beyond the analyzed set, it is conceivable that some systems would have mutually closing control loops without shared authority.

The plot also suggests that both *dynamic hierarchy* and *dynamic authority* only occur in the set when the interaction also involves *cognitive alignment*. Both of these dynamics require the team to resolve which controller is in control of part of the process or which controller is in control of other controllers over time. As such, it intuitively makes sense that the controllers would need to align their models and decisions accordingly.

Similarly, *mutually closing control loops* was found only in the presence of *cognitive alignment*. This correlation may reflect the system-theoretic foundation of this work. In mutually closing control loops, feedback loops to the process are closed through multiple controllers. If those controllers were abstracted as one controller as a whole, that controller would need a process model to control the process. As such, in a refined view of the system, the multiple controllers involved in the feedback loop must have cognitive alignment to safely close those loops.

Despite the proposed explanations for these correlations, examples may exist beyond the systems studied for which the relationships do not hold true. No definitive conclusions can be drawn on the six correlations between dynamics based on the results of this study alone. However, the knowledge of these potential implications can help steer an analyst or a designer to look out for certain collaborative dynamics if others have been identified.

3.4 Summary of Collaborative Control Definition

Current systems engineering processes are challenged by some of the more complex team-inspired component interactions sought in aerospace system designs. As such, the community is unable to properly model, analyze, and design for assurance of such systems. This research aims to overcome some of these limitations by developing a rigorous and systematic technique to analyze the safety of multi-controller collaborative systems.

As a first step to accomplishing this goal, this chapter defined the diverse types of collaborative relationships that multi-controller teams may exhibit. It introduced a system-theoretic framework to describe different types of interactions that are—or are planned to be—designed into aerospace systems. A taxonomy helps reason about seven different structural dimensions in controller interactions. In addition, a set of nine dynamics observed in collaborative control were defined using Systems Theory and are grounded in STAMP.

To determine whether and how this taxonomy relates to fielded and proposed systems, a set of 101 system interactions studied in this work were categorized using the framework. The associated analysis demonstrated that each of the collaborative control dynamics is found in system concepts the aerospace community wants to field. It also showed that these dynamics are more prevalent in conceptual systems than in those already fielded. The results support Hypothesis 1 of this dissertation, which underlies the need for the framework.

> *Hypothesis 1: The system-theoretic collaborative interactions framework provides a mechanism to categorize and describe component interactions that are, or planned to be, designed into aerospace systems.*

No claim is made that the taxonomy or the set of collaborative control dynamics is complete. There may be additional dimensions that are determined to be important in some other applications, and additional collaborative control dynamics of interest may be uncovered. Furthermore, the framework developed in this chapter can be employed to explore other types of interactions between controllers, beyond collaboration. This is the subject of future work.

The system-theoretic definition of collaborative control dynamics presented in this chapter provides the necessary foundation to incorporate these relationships into the underlying causality model of STAMP-based techniques. The extensions to STAMP and STPA developed to systematically identify causal factors associated with these interactions are developed in the next chapter.

Chapter 4: Extending STAMP and STPA for Collaborative Control

The system-theoretic foundation of STAMP and STPA is well-suited to address some of the challenges in modeling and analyzing the novel aerospace systems introduced in Chapter 1.1. However, these methods need additional guidance to clarify how to handle causality associated with the more complex team-inspired component interactions sought in new designs. While STPA may find some causal factors associated with collaborative control, it is vulnerable to missing others because it lacks a systematic approach to address such relationships.

This problem was evident when analyzing the safety of a future helicopter concept that executes missions *optionally-manned*[5] and in collaboration with multiple UAS [28]. The system was modeled with a control structure that includes a "teaming controller" to help STPA explore teaming interactions (Figure 4-1). The function of the teaming controller is to dynamically coordinate resources in the multi-aircraft team to address mission needs. The teaming controller could be implemented as a centralized, distributed, or hybrid controller.

The model did help identify several causal scenarios that resulted from the breakdown of collaborative control among the aircraft. The results include examples of conflicting control, unsafe control handoffs, human-machine trust issues, inconsistent semantics in the team, incompatible system configurations, and more [28]. While these results are good and useful, several challenges were encountered during the analysis.

For instance, the causal relationships for many of the unsafe collaborative control issues listed above are not expressed in the control structure. The results were only obtained by analyzing the model creatively, through the lens of teaming, and drawing on past expertise in collaborative control. Earlier versions of the analysis, without that perspective, did not consider the issues listed above. For a technique to be repeatable by different analysis teams, the causal influences that occur in collaboration must be explicitly accounted for by the model and analyzable using a systematic process.

The teaming controller also led to ambiguity in the hierarchy of components in the model. For example, by controlling team resources, which include the operator's aircraft, the teaming controller may issue control actions to the operator and therefore be hierarchically superior. However, the reverse also occurs when the operator directs the actions for the team, as represented in the control structure.

Challenges also occurred in abstracting the model to a higher level. It is ambiguous if the teaming controller belongs to the operator, the aircraft software controller, the UAS, or all three

[5] *Optionally-manned* means a human pilot may or may not be physically onboard the aircraft. If no pilot is onboard, the aircraft receives piloting commands remotely.

if distributed. Similarly, it is unclear how a higher-level model of such a distributed system would be consistently refined to produce the representation in Figure 4-1.

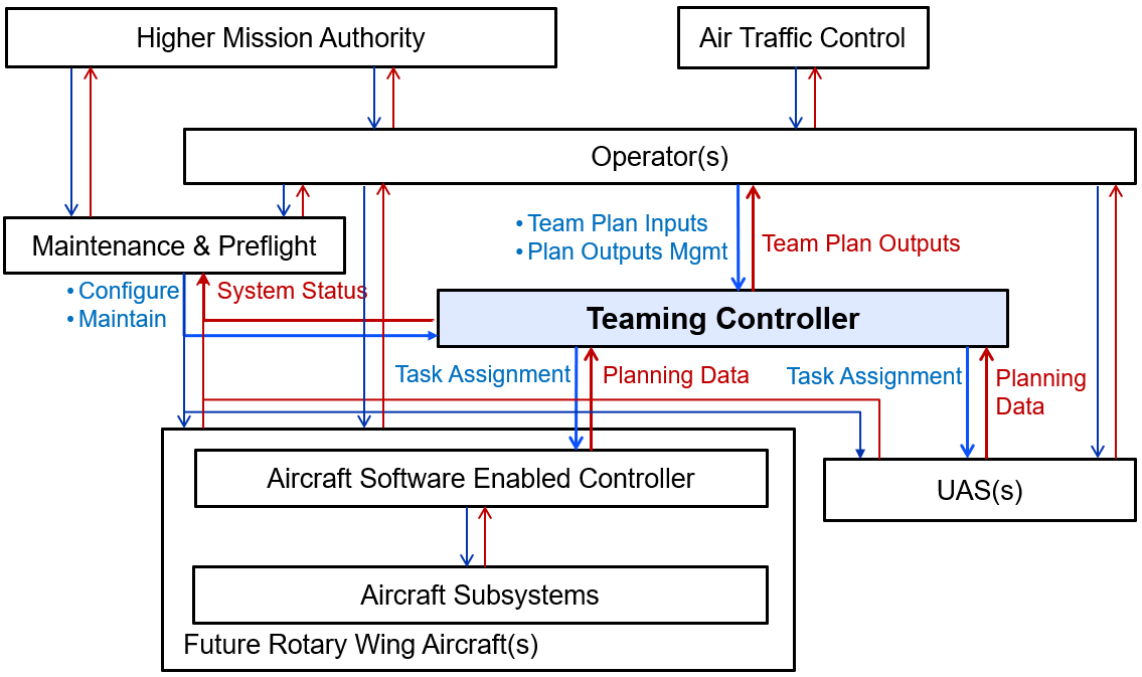


Figure 4-1: "Teaming Controller" in Future Helicopter Control Structure (adapted from [28])

Despite the useful results obtained with the teaming controller as represented, the lack of a systematic approach to handle collaborative control could have resulted in the system being modeled differently. This variation can lead to different results in the hazard analysis and increases the risk of missing causal factors. As such, further guidance is needed in STAMP and STPA to handle these situations more consistently.

This chapter introduces extensions to STAMP and STPA to add guidance and rigor in the analysis of collaborative control interactions. The new methods are grounded in the collaborative control framework defined in Chapter 3. They are also derived from the existing guidance in STAMP and STPA so that they remain consistent with these proven techniques. The following is an overview of the three extensions, which are illustrated in Figure 4-2 and are collectively referred to as *STPA-Teaming*.

First, a *generic collaborative control structure* is developed to incorporate the types of interactions exhibited in teaming into STAMP models. It serves as a reconfigurable template to assist in modeling the relationships between multiple humans and/or machines that collaborate in the control of a process. The goal is to explicitly express these mutual influences so that causal factors associated with them can later be systematically identified.

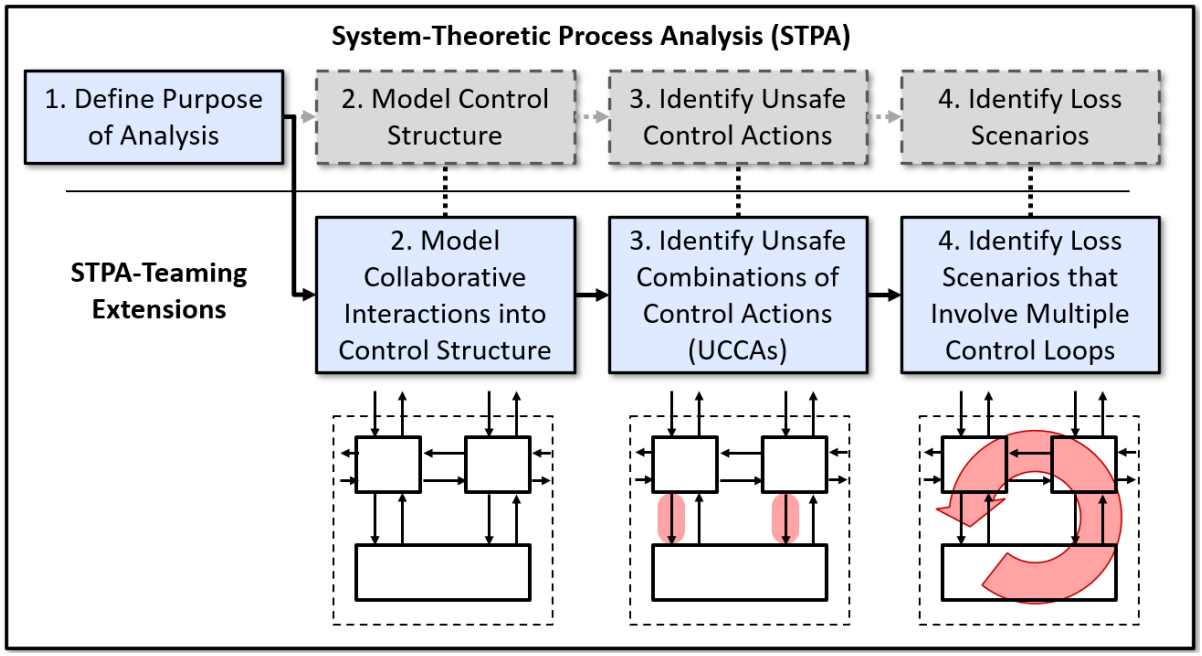


Figure 4-2. Three Analytical Extensions Involved in STPA-Teaming

Second, the process to identify unsafe control actions (UCAs) in STPA is expanded to explore how combinations of multiple control actions provided together may lead to hazards. The actions of multiple controllers are systematically analyzed relative to one another using an approach derived from the four *types of UCAs* defined in STPA [50]. A method of abstracting and refining the control structure helps manage combinatorial complexity in this process. Finally, a prototype automation tool is introduced to support an analyst with enumerations, refinement, and prioritization of the unsafe combinations of control actions to analyze. The overall procedure systematically considers potential issues involving control gaps, overlaps, transfers, and controller-task mismatches that are found in collaborative control.

Third, an analytical procedure is introduced to develop causal scenarios that explain how these unsafe combinations of control actions (UCCAs) could occur. It follows a structured search process inspired by Thomas [203] and is framed by the collaborative control dynamics defined in Chapter 3. The method emphasizes defining scenarios at a high level and refining them iteratively, as necessary, and guided by the types of interactions between controllers.

The remainder of the Chapter provides details associated with the development of each of these extensions. These techniques are then demonstrated in a case study on a real-world aerospace system concept in Chapter 5.

4.1 Generic Collaborative Control Structure

STAMP hierarchal control structures provide a powerful mechanism to model complex systems holistically and top-down. The ability to represent causal relationships between components at multiple levels of the system is one of the reasons why STAMP is so successful at finding factors

that threaten safety. In recent years, several efforts have aimed to enhance STAMP models so that they produce more complete analyses.

The synthesis of these efforts helped define the generic control structure shown in Figure 4-3 and found in Appendix G of the STPA Handbook [50]. This reference also explains how each element of the model can contribute to causal scenarios found using STPA. The model consists of a human operator that supervises automation, which controls a process. It is representative of many supervisory control systems currently fielded.

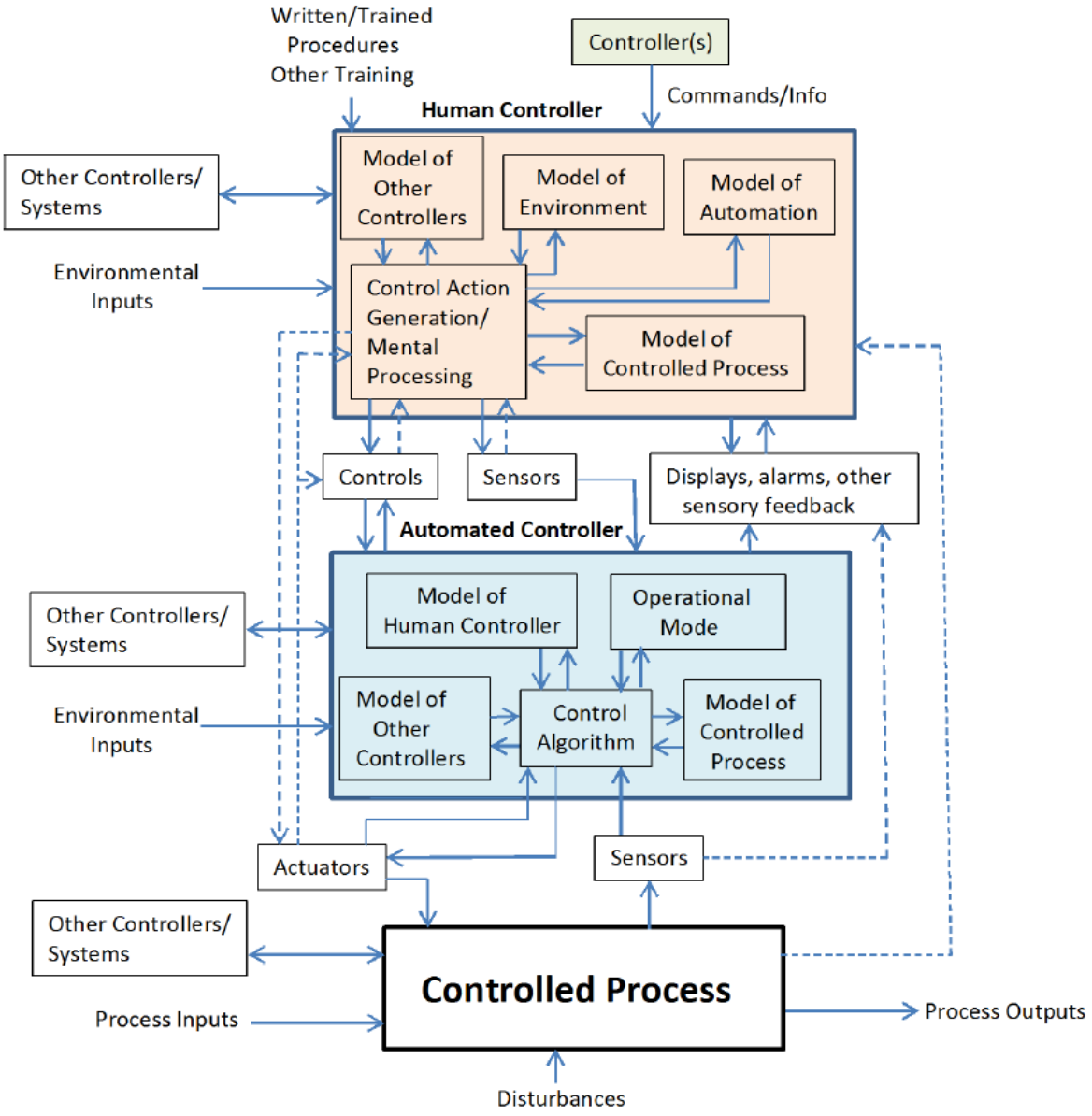


Figure 4-3. Baseline Generic Control Structure from STPA Handbook Appendix G [50]

Unfortunately, the collaborative control dynamics defined in Chapter 3 are not explicitly represented in this model, nor in other existing control structures. This increases the risk that those causal relationships will either be missed or will not be systematically handled in the ensuing hazard analysis. To address this gap, this dissertation introduces the *Generic Collaborative*

Control Structure shown in Figure 4-4 as a reconfigurable template to represent various teaming systems. The model is grounded in STAMP, but it also extends the available guidance to incorporate collaborative interactions.

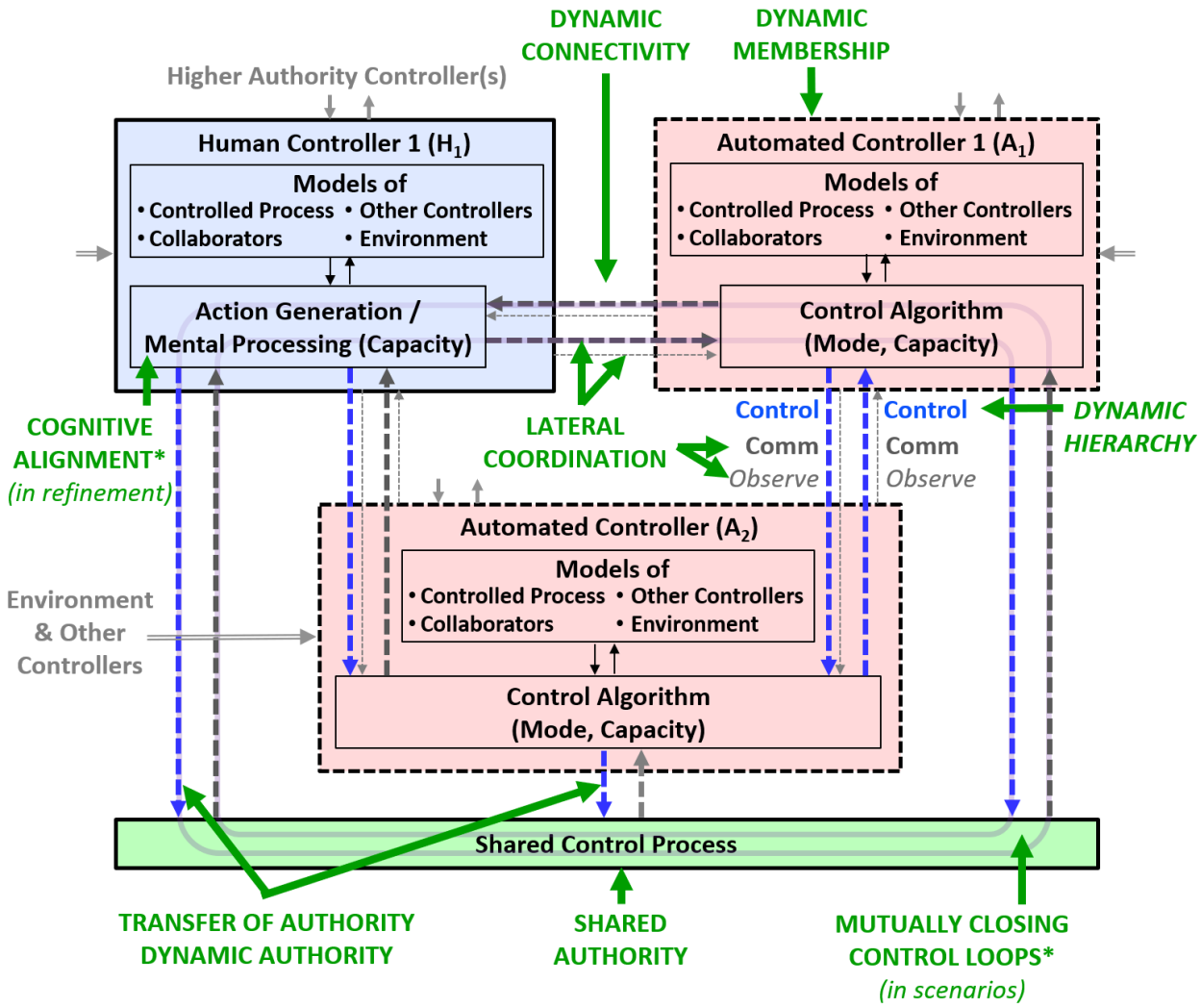


Figure 4-4. Generic Collaborative Control Structure

The remainder of this section is organized as follows. Section 4.1.1 provides an overview of the model and relates it to STAMP. Section 4.1.2 describes the cognitive functions of the controllers in the model, which underpin several of the collaborative control dynamics. Section 4.1.3 explains how the collaborative control dynamics are integrated into the control structure. Finally, Section 4.1.4 provides additional modeling recommendations.

4.1.1 Overview of the Model

The generic collaborative control structure builds on the baseline in Figure 4-3 to express collaborative interactions while remaining consistent with STAMP. Every element described in the baseline applies to the extension. However, abstraction allows some of the details to be hidden so that other features more aligned with the research focus on collaboration can be

highlighted. For example, the *actuators* in Figure 4-3 are abstracted away from the control paths, as are the *sensors* in the feedback paths. These lower-level components are still part of the collaborative control model, and if needed can be handled using the guidance from the baseline.

Each controller in the extended model includes the same high-level cognitive functions described in the baseline. The relationship of these functions to collaborative control is further discussed in Section 4.1.2. The interactions the controllers have with higher authority controllers, with other controllers (beyond the set of collaborators), and with the environment are also shown and are handled no differently than they are in STPA.

The generic collaborative control structure shown in Figure 4-4 represents one of any arbitrary collaborative system configurations. In this case, it includes a human controller (H_1) working as a peer with an automated controller (A_1). Together, H_1 and A_1 have authority over another automated controller (A_2). All three controllers collaborate in controlling a shared process.

These building blocks can be reorganized into any other configurations. For instance, Figure 4-5 shows the collaborative control structures of different systems in development for Urban Air Mobility (UAM). These concepts involve different potential architectures for human and/or automated controller collaborations [8], [10], [204]. The *authority bus* in the figure indicates shared authority by the controllers over the subprocesses and is further discussed in Section 4.1.4.

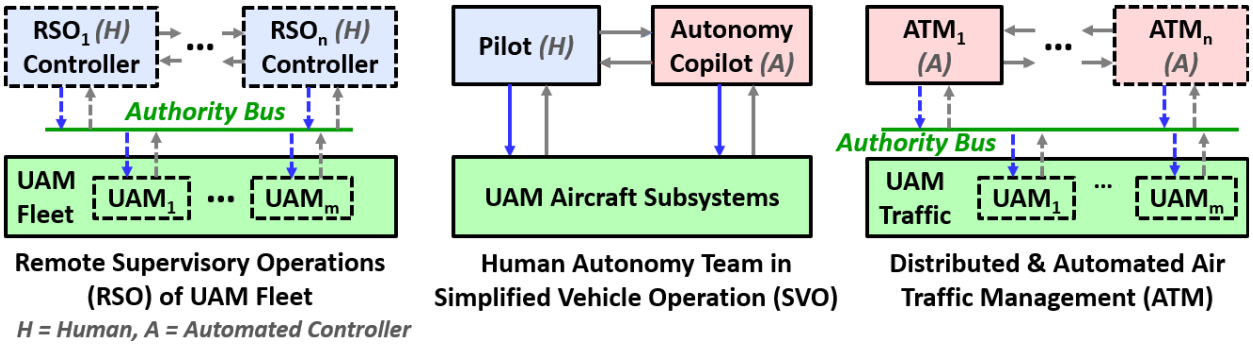


Figure 4-5. System Concepts Demonstrating Various Collaborative Control Configurations

The system-theoretic framework allows collaboration to be represented at multiple levels of abstraction and hierarchy. For example, while Figure 4-4 shows all three controllers collaborating to control the shared process, it can also be analyzed as H_1 and A_1 collaborating in the shared control of A_2. Similarly, the human-machine system modeled could be abstracted as a whole, and work collaboratively with other human-machine systems on a shared process. The interactions highlighted in the collaborative control structure can be applied to any set of collaborators sharing a process. Strategies to navigate between these different views are presented in Chapter 6.

4.1.2 Cognitive Functions

The baseline causal model (Figure 4-3) defines two sets of high-level cognitive functions for each controller [50]. The first processes information to make control decisions. The second consists of various models that support the decision-making process. STPA provides guidance to consider how a controller may have flaws in these functions that contribute to its unsafe control actions.

The collaborative control structure carries over these concepts, but it emphasizes instead how these functions can be flawed relative to one another across multiple collaborating controllers. Figure 4-6 provides an overview of how cognitive functions are integrated into the extended model. The following discussion explains these functions and how they were derived from the baseline guidance and other references. These processes are highly relevant to many of the collaborative control interactions further discussed in Section 4.1.3.

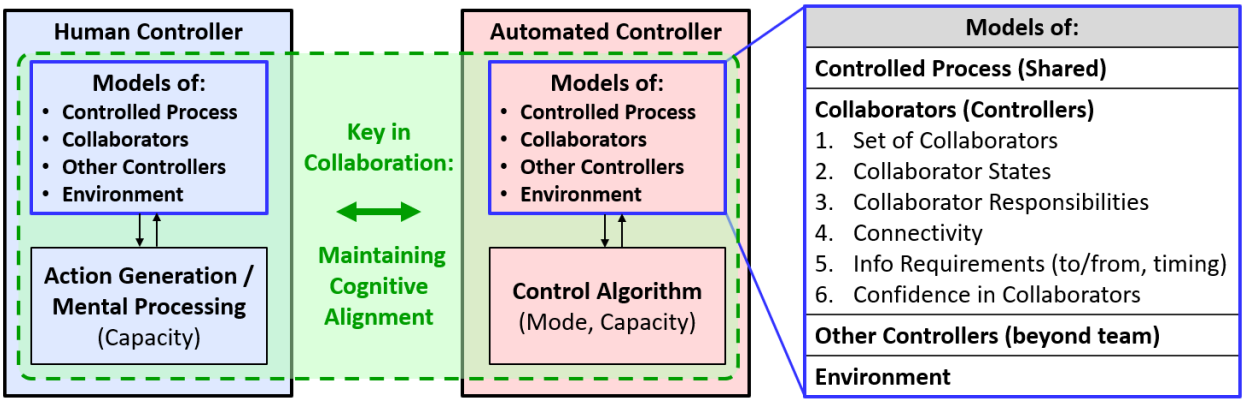


Figure 4-6. Cognitive Functions in Collaborative Control Structure

Information Processing and Decision Making

As explained in the baseline, automated controllers have a *control algorithm* that processes inputs to the controller to (1) generate control actions, and (2) maintain accurate information about the state of the system by interacting with the controller models. The behavior of the control algorithm is shaped by the operational modes of the system [50].

The extended model integrates all these components. One subtle difference is that the control algorithm is emphasized to generate *actions*, which include *control actions* as in the baseline, but it also consists of other *communication actions* to coordinate with and influence the behavior of other controllers without using control. For example, the Aircraft Collision Avoidance System (ACAS-X) system described in Chapter 3.3 allows the control algorithm of a lower-ranked aircraft to output a deconfliction solution first, which influences the decision of the higher-ranked aircraft.

Another consideration added to the model is the limitations placed on the control algorithm due to controller *capacity*. In Johnson's work, limited capacity is a key factor in determining how controllers form dependencies on one another to execute joint activities [2]. He broadly defines capacity as the set of knowledge, skills, abilities, and resources a controller needs to perform an action. Some of these elements are already explicitly considered in STAMP in the form of control paths, feedback paths, and models. However, other aspects require attention and are important in collaborative interactions.

In this work, the capacity of a control algorithm refers to the factors that can limit its ability to track and update models and select appropriate output actions. Limited internal resources, including computational, data, or communication, may prevent a controller under a certain workload to output the behavior that was expected by its collaborators. Such issues can cause a misalignment in cognition across multiple controllers and contribute to an unsafe team output.

These concepts also apply to the human controllers in the extended model. The baseline represents the human with a cognitive function for *control action generation & mental processing* [50]. While its purpose is comparable to that of the *control algorithm* in machines, the STPA guidance emphasizes that humans are more complex and are subject to different causal factors.

The collaborative control structure similarly broadens the scope of the function to *action generation & mental processing* so that it accounts for the coordination outputs in addition to the control actions. Humans working collaboratively often deliberately provide coordination actions to influence the behavior of others they cannot control. For example, flight crews often verbalize hints to each other regarding recommended actions.

Similarly, humans are also subject to *capacity* limitations in their ability, skill, or workload that influence how well they maintain situational awareness and make decisions. For instance, an inexperienced pilot who is still learning to process information efficiently will more easily be overwhelmed, "fall behind the airplane", and make bad decisions.

Model of the Process

The cognitive function described above updates, maintains, and relies on multiple models of the system to select the actions to generate [50]. The purpose of the models is similar between humans and automated controllers, but the reasons for their flaws can vary greatly. As with other baseline STPA elements, these models are integrated into the generic collaborative control structure and related to the process of collaboration.

In STAMP, the model of the process represents the state that the controller believes the process to be in. The controller relies on this model to select control actions that will constrain the behavior of the process so that it does not enter a hazardous state [38]. The STPA guidance provides many reasons why an automated controller may have a flawed process model. These include inadequate feedback from the process, delays in receiving or processing the feedback, and flawed assumptions based on control inputs. Human controllers may, in addition, be subject to mode confusion, lack of situational awareness, confusion due to lack of transparency, or even complacency [50]. These all apply to the collaborative control structure.

However, collaborative control brings on additional considerations as the *control process is shared* between multiple controllers. It is not sufficient to just consider what state one controller believes the process to be in. The consistency of the process models across multiple controllers is critical. As reviewed in Chapter 2.1, teams rely on both shared and distributed cognition.

Shared cognition relates to information held in common between multiple controllers [60]. If information is misaligned, the controllers may have different beliefs regarding the process state and may issue commands that are inconsistent with one another. This was likely a contributing factor in the Air France Flight 447 accident (see Chapter 1.2), when the two pilots had misaligned models about the state of the aircraft, leading one to pitch up and the other to pitch down [35].

Shared cognition may also cause a controller that has a valid model of the process to drift toward incorrect beliefs provided by a collaborator. This can occur in the psychological phenomenon of *groupthink*. In addition, a controller that has a flawed process model may receive negative reinforcement from the flawed model of a collaborator. This can also occur in *groupthink* and in *confirmation bias*. As such, the model flaws in one controller can propagate to others.

Distributed cognition focuses on differences in knowledge between collaborators and emphasizes the need to coordinate around who knows what and when on the team. The key implication of this concept is that in collaboration, any one controller may not have direct access to the complete state of the process to decide what actions to take. It may rely on other controllers to receive the necessary feedback to inform its actions.

Flawed distributed cognition contributed to the 1994 friendly fire shootdown of two U.S. Army helicopters (see Chapter 1.2). Two combat air traffic controllers had split responsibilities to respectively maintain mental models of aircraft within and outside a prescribed area. The responsibility to track low-flying helicopters evolved over time and was eventually left unassigned for particular situations. In the accident, the controller for inside the area relied on an incomplete model from the other controller and, as a result, made an unsafe decision [38].

Model of the Collaborating Controllers

The STPA guidance also describes how a controller may have a model of other controllers it interacts with. For example, a human controller must have a model of the automation to supervise its control of the process. Similarly, some sophisticated automated controllers have models of the humans that are supervising them [50]. In this work, the generic collaborative control structure incorporates this concept as a *model of collaborators*.

Several information items may form the model of collaborators. Examples found in the systems sampled in Appendix 1 are illustrated on the right of Figure 4-6. While the content of these models can vary widely, the intent of the figure is to help reason about the type of information controllers track about their teammates to shape their individual output decisions.

The model of collaborators may include knowledge of the set of collaborators involved in the activity. This becomes particularly important in teams that exhibit dynamic membership when this set changes over time. A controller may exhibit unsafe behavior if it is not aware that it has a teammate or if it falsely believes that it does.

Some systems require controllers to track the state of their collaborators, such as their location and trajectory. For example, in implicit coordination algorithms, each controller uses the state information for all members of the team to compute a plan for the whole team and execute their portion of the plan. If the state information and the algorithms are consistent across controllers, they can produce safe, coordinated solutions [83].

In some cases, teammates may be required to track the responsibilities of other collaborators. Such information can be necessary for collaborative systems that exhibit dynamic authority, in which controllers determine the allocation of control during execution. In some coordination schemes, such as market-based algorithms, teammates form consensus over their responsibilities only and do not have to rely on other state information [83].

Controllers on a team may need to estimate the network topology. Many systems exhibit dynamic connectivity, in which communication channels between controllers are expected to vary. The knowledge of which controllers can be reached at any time, either directly or indirectly, may influence how a control decides to output coordination and control actions.

Beyond connectivity, controllers may track information requirements between controllers. In distributed cognition, controllers must understand what information they need from others and what information others need from them [39]. The timing requirements for information sharing

may also vary based on the context and the types of controllers involved. There are many examples of automated controllers that unduly interrupt the workflow of humans, or that update information too quickly for humans to process [23].

Finally, controllers may assess their confidence in the behavior of other controllers. A controller that has no confidence in the output of a teammate may choose a different action than if it did. Asymmetric assessments of confidence lead to misaligned decisions in joint control.

Models of Other Controllers and the Environment

As described for STPA, controllers can rely on additional models in selecting their actions. They may have a *model of other controllers* involved in the system. In this work, these are controllers beyond the set of *collaborative controllers* considered above. These other controllers may interact with the team, the process, or some other part of the system in a different way. Similarly, the controllers may also maintain models of the environment, or components beyond the system boundary that the system interacts with. In this work, these models are as they are in STPA [50].

4.1.3 Collaborative Control Dynamics in the Control Structure

A key goal of the extended control structure is to express the collaborative control dynamics defined in Chapter 3 that are exhibited by the system. This section explains how this is accomplished using the items labeled with the green arrows in Figure 4-4.

A key consideration in collaborative control is *shared authority*. It is expressed in the control structure with the *shared controlled process*, which is the joint control activity over which multiple controllers have authority. It may represent a set of mission tasks, a formation, trajectory deconfliction, or any other process that is jointly controlled.

Many past STPA studies relevant to teaming, including the one referenced in Figure 4-1, do not show this process in the control structure, and instead, only list the various controllers that collaborate. The inclusion of a shared process is necessary to reason about how combinations of control actions by multiple controllers may be hazardous. This concept is central to the Unsafe Combination of Control Action (UCCA) identification technique introduced in Section 4.2.

The use of dashed lines in the collaborative control structure symbolizes the dynamic presence of an item. This convention is repeated in multiple ways. The dashed control and feedback arrows that lead to and from the shared process indicate *dynamic authority* or *transfer of authority*. The significance in both is that the controller from which the dashed control line originates may not always be responsible for issuing the control action.

In dynamic authority, the control path is allocated, or possibly reallocated multiple times, to one of the collaborative controllers during execution. In transfer of authority, the control path is handed off from one controller to another over time. Control gaps, overlaps, and mismatches that can arise from these dynamics may lead the system into a hazardous state. These two concepts are also inherently captured by the UCCA extension described in Section 4.2.

The arrows between the controllers are dashed to symbolize *dynamic connectivity*. Those connections may or may not be available at any given time. Similarly, some of the controllers have dashed frames to indicate *dynamic membership*. This means that those controllers are not

always part of the control structure. The causal implications of dynamic connectivity and dynamic membership, as well as those for all the remaining collaborative control interactions, are covered in the scenario identification process in Section 4.3.

In STPA, items listed on downward arrows from one controller to another are typically treated as control actions and analyzed for UCAs. The arrows pointing up are feedback items and those connecting controllers laterally consist of other information [50]. This convention is relaxed in the extended control structure to account for additional interactions in collaboration. While the control structure is still organized hierarchically, not every item flowing down is necessarily a control action, and control actions may be included on lateral and upward arrows.

In Figure 4-4, each connection from one controller to another includes two arrows. One arrow, shown in bold, originates from the *action generation* cognitive function. The other arrow, not bold, stems instead from the overall controller frame. Both arrows terminate at the cognitive function of a receiving controller as inputs to inform its decisions.

The bold arrow flowing out of the cognitive function reflects information deliberately provided by the controller to influence another controller. It consists of control actions (bold and blue) and communication actions to enable lateral coordination (bold not blue). To clarify authority, if the arrow has a control action, then it is shown in blue even if it also includes communication items. In the ensuing hazard analysis, control actions are analyzed for UCCAs (see Section 4.2) and communication actions are considered in causal scenario development (see Section 4.3).

The thinner arrow, which does not originate from the cognitive function, represents the information obtained by observing a controller, as part of lateral coordination. As described in Chapter 3.2, even though this information is not deliberately provided to coordinate, it can implicitly influence the decisions of collaborators. Observation items are also considered in the scenario development process (see Section 4.3).

In Figure 4-4, both A_1 and A_2 can provide control actions to one another. This is indicative of *dynamic hierarchy*, in which a controller leads part of the interaction, and another controller leads in another part. This dynamic is also captured in the scenario development process.

The extended model shows how *mutually closing control loops* can be identified between multiple controllers and the shared process. However, this dynamic is further explored in scenario development using a more focused control structure for the control loops being analyzed. Similarly, a label for cognitive alignment is also shown in Figure 4-4, but it is more closely considered by accounting for the cognitive functions of multiple controllers in scenario development. These topics are detailed in Section 4.3.

4.1.4 Additional Recommendations for the Model

The following additional recommendations can help analysts model collaborative systems. In some cases, it may be easier to include an *authority bus* to represent shared authority over multiple processes, as shown in Figure 4-5. The bus indicates that all the controllers that feed into it can have authority over the processes that receive an output from it.

In addition, an indexing scheme of $\{1, \ldots, n\}$ controllers, also shown in Figure 4-5, helps account for a variable number of similar types of controllers. In such cases, it is recommended that at least two controllers be shown so that the interactions between them can be expressed and considered in causal analysis. Including more than two similar controllers is often not necessary and adds complexity to the analysis.

4.2 Unsafe Combinations of Control Actions (UCCAs)

STPA employs a systematic method derived from Control Theory to identify unsafe control actions (UCAs). The method has been shown to be complete in its ability to describe how a particular command may be provided in an unsafe way [50], [205]. However, this process often lends itself to considering one controller and one of its feedback control loops at a time.

Collaborative control fundamentally involves multiple controllers working together to control a shared process. A key implication of *shared authority*, as defined in Chapter 3, is that the control actions from collaborating controllers cannot be analyzed individually. There may be unsafe causal factors that can only be identified when the actions of these multiple controllers are analyzed together.

For example, a flight crew may involve a pilot that controls aircraft attitude and trajectory using the flight control yoke and throttle, and a second pilot that controls aircraft configuration by selecting flaps and landing gear settings. Configuration changes alter the aerodynamic properties of the aircraft and therefore alter how attitude and trajectory are controlled. Similarly, variations in attitude and trajectory may cause the aircraft to enter an operating state that necessitates a configuration change. The control actions of each controller may be unsafe in the context of the actions provided by the other controller.

Collaborative control may also involve *dynamic authority* that enables multiple controllers to adjust task allocation during execution (see Chapter 3). However, that may result in control gaps, conflict in overlaps, or controller-task mismatches. Similarly, a system may allow *transfer of authority,* which can lead to unsafe control handoffs. These types of relationships were exhibited by the operator teams involved in the infamous Air France 447 and 1994 Friendly Fire accidents described in Chapter 1.

In STPA, a human analyst specifies the context in which a control action is unsafe [50]. Suggestions are provided in the STPA Handbook for how to do this. Thomas provides formalism to identify context using process variables derived from the system hazards [205]. However, these methods do not explicitly define how to explore a control action that may be unsafe relative to other commands.

Placke's work provides a useful first step to address this gap. His approach explores how one controller may interfere with another [206]. Specifically, he defines *Conflict UCAs* of the form: <*Controller 1 provides Command A*> prior to <*Controller 2 provides Command B*> violates <*design assumptions*>. However, the method does not address different types of multi-controller interactions, such as control gaps, unsafe handoffs, and dynamic tasking. Furthermore, the analysis is limited to pairwise comparisons of two control actions. Finally, the formulation only

relates to a subset of the four Types of UCAs defined in STPA. This means that there are other ways to issue control actions relative to one another that must be considered.

This section introduces an extension to STPA to explore how multiple control actions may be unsafe together. The process identifies Unsafe Combinations of Control Actions (UCCAs) and has the following attributes. First, it follows a systematic approach derived from STPA to ensure all relevant types of control combinations are considered. Second, it leverages multiple levels of abstraction to manage combinatorial complexity. And finally, the UCCAs enable the analysis of their causal factors to be framed by the collaborative control dynamics defined in Chapter 3.

The remainder of this section is organized as follows. Section 4.2.1 derives the different types of UCCAs and provides the foundation necessary to enumerate them algorithmically. Section 4.2.2 introduces an initial framework of abstraction to reduce the combinatorial complexity of the enumeration. Section 4.2.3 adds further abstraction to linearize the growth of UCCAs for more complex systems. Finally, Section 4.2.4 integrates these concepts into an algorithm to identify, refine, and prioritize UCCAs for their follow-on causal analysis. A prototype tool built on the formalism of this algorithm supports its execution by automating some of the steps.

4.2.1 Types of UCCAs

One of the strengths of STPA is its systematic process of considering how a control action, given a particular context, can lead to a hazardous state. UCAs are identified using the specific structure shown in Figure 4-7 [50].

	1	2	3	4	5
UCA Structure:	<Controller>	**<UCA Type>**	<Control Action>	<*Context*>	*[H]*

Figure 4-7. Structure of a UCA in STPA

The controller (item 1) and the control action (item 3) are obtained from the control structure. STPA defines four types of UCAs (item 2), listed below, in which a control action may be unsafe. The set of four types is provably complete to describe a given control effort [50].

- UCA Type 1: not providing the control action
- UCA Type 2: providing the control action
- UCA Type 3: providing the control action too early or too late
- UCA Type 4: providing the control action for too long or stopping it too soon

The structure in Figure 4-7 allows items 1-3 to be machine enumerated. This reduces the burden on a human analyst, who can instead focus on determining if there is a context for the UCA (item 4), and, if so, trace it to the hazard(s) it leads to (item 5). Examples of each type of UCA are provided in Table 2-3.

The process to identify Unsafe Combinations of Control Actions (UCCAs) builds on the approach for UCAs. The UCA structure is expanded to incorporate combinations of control actions. Different types of UCCAs are derived from the types of UCAs to maintain the rigor provided by STPA. The resulting formulation provides the foundation necessary to enumerate combinations of control actions algorithmically. As such, the potential UCCAs can be machine-

generated and, again, focus the human analyst on specifying the context in which the control combinations are unsafe.

Two UCCA types are defined. The first, UCCA Type 1-2, combines UCA Types 1 and 2, which describe whether or not a control action is provided at all. Type 1-2 UCCAs help find contexts in which providing none, some, or all of multiple control actions is unsafe. As a simple example, consider two controllers c_i and c_j, that can each provide control action u_a and u_b respectively. Potential Type 1-2 UCCAs can be enumerated as follows, using an extended UCA structure with the same color coding as in Figure 4-7:

Type 1-2 UCCA Example:

1. c_i **does not provide** u_a and c_j **does not provide** u_b when… *[H]*
2. c_i **does not provide** u_a and c_j **provides** u_b when… *[H]*
3. c_i **provides** u_a and c_j **does not provide** u_b when… *[H]*
4. c_i **provides** u_a and c_j **provides** u_b when… *[H]*

Second, the Type 3-4 UCCA combines UCA Types 3 and 4, which assume that the control action is provided, and instead focuses on the temporal sequence in which it occurs. Specifically, a Type 3 UCA considers when the control effort starts, rising from OFF to ON, as shown with a step function in Figure 4-8. A Type 4 UCA accounts for when the control effort ends, falling from ON to OFF. For a discrete command not provided over time, only the rising edge is considered, and Type 4 UCAs do not apply [50].

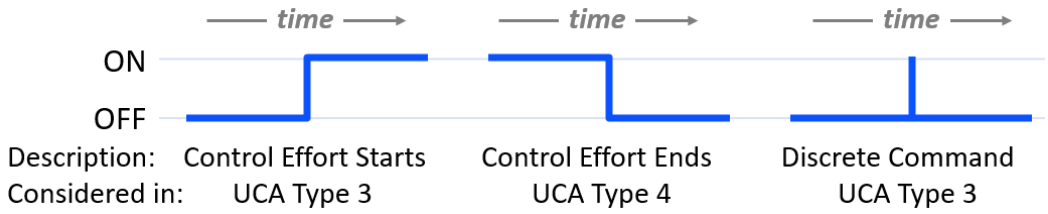


Figure 4-8. Start and End of a Control Effort Considered in UCA Type 3 and 4

As such, Type 3-4 UCCAs explore the temporal sequences of multiple control actions in relation to each other that are unsafe. Specifically, they analyze how starting or ending certain control actions before or after starting or ending other control actions may be unsafe. The Type 3-4 UCCAs are enumerated using the extended UCA structure in Figure 4-9 for the same example as above.

To maintain generality, no assumption is made that a control action, if started first, has not ended before the second action starts or ends. This subtlety accounts for discrete commands, which are not applied over time, as shown in Figure 4-9. The same assumption also allows the first command, if continuous, to be started and ended before the second action starts or ends. Similarly, the first command can be ended and then started again before the second command changes.

To remain consistent with UCA Type 4 in STPA, the end of a discrete command is not considered[6]. As such, if u_a is discrete, enumerations {3,4,7,8} in Figure 4-9 are not applicable. Similarly, if u_b is discrete, enumerations {2,4,6,8} are excluded. If both are discrete, only items {1,5} apply.

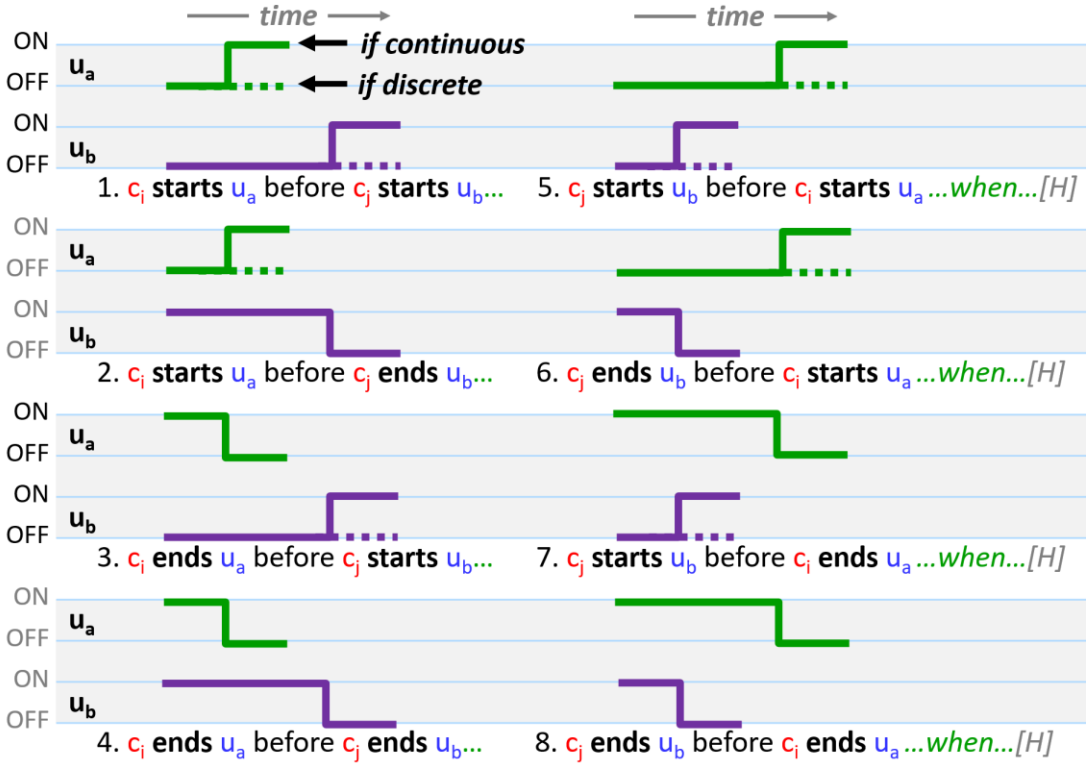


Figure 4-9. Type 3-4 UCCA Example

The two types of UCCAs provide the foundation for a machine to enumerate all possible control combinations. However, that does not by itself solve the problem of analyzing how multiple control actions are unsafe together. The example above consists of a simple pairwise comparison of two control actions. Additional processes, introduced in the next subsections, are needed to handle more complex combinations.

4.2.2 Managing Combinatorial Complexity with Abstraction

This section introduces a method to systematically manage the combinatorial complexity that occurs when enumerating UCCAs. First, a simple example illustrates the need for the process. Next, a process of abstraction is developed to solve the problem.

[6] This defines discrete commands as different than continuous commands. As such, a discrete command cannot be treated as, or converted to, a continuous command in later steps of the hazard analysis. If a discrete command must be changed to a continuous command, the Type 4 UCA and Type 3-4 UCCA identification must be re-addressed with this change.

Combinatorial Complexity of UCCA Enumeration

Consider the following multi-UAS system concept used in the STPA example in Chapter 2.4 [143]. An operator controls two collaborative UAS by specifying mission tasks for the collective team to perform. These tasks represent control actions the UAS must provide to the shared mission process, such as *jamming* a radar and *striking* a target. In this example, the system enters a hazardous state if the team strikes the target without jamming the radar or if multiple UAS jam the radar simultaneously.

Figure 4-10 shows the control structure for this simple example. The two UAS can provide both the *jam* and the *strike* commands. They coordinate with each other to determine which UAS will perform each task. The goal in this work is to systematically explore how combinations of UAS_1 and UAS_2 issuing the *jam* and *strike* commands may be unsafe.

Figure 4-10 also includes a generalized form of the model, with $n = 2$ controllers that can each issue the $m = 2$ different control actions. This representation is referenced in the algorithmic enumeration discussion below.

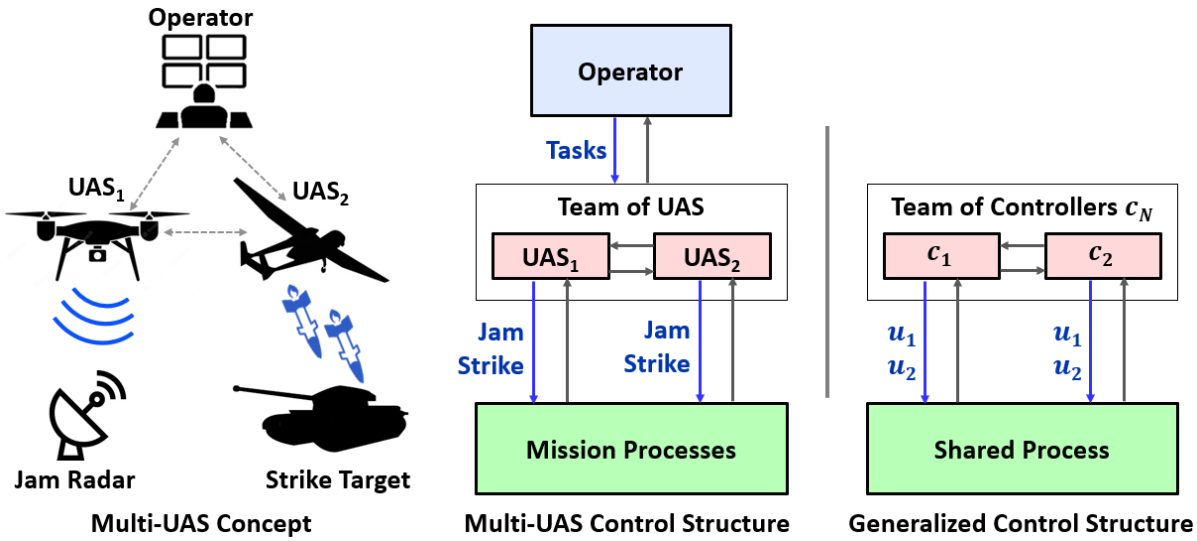


Figure 4-10. Illustrative Multi-UAS Team Example (left) and its Generalized Form (right)

The Types of UCCAs defined in the previous section help enumerate all the combinations of control actions provided by the two controllers in this example. Using the extended UCA structure, Type 1-2 UCCAs are formulated as:

1. c_1 **does not provide** $\{u_1$ *or* $u_2\}$; c_2 **does not provide** $\{u_1$ *or* $u_2\}$ when… *[H]*
2. c_1 **does not provide** $\{u_1$ *or* $u_2\}$; c_2 **does not provide** u_1 and **provides** u_2 when… *[H]*
3. c_1 **does not provide** $\{u_1$ *or* $u_2\}$; c_2 **provides** u_1 and **does not provide** u_2 when… *[H]*
4. …

It is simpler to enumerate combinations using a format inspired by truth tables, as shown in Table 4-1. The top row designates the controllers that issue the control actions in each subsequent row. Here, $\neg u$ means "does not provide u". Rows 1-3 in the table match one-to-one the three UCCAs specified in English above.

In this relatively simple problem, there are already 16 potential Type 1-2 UCCAs for the analyst to evaluate. There are even more potential Type 3-4 UCCAs. As shown in Figure 4-9, there are 8 ways to order the start and end of two different control actions. This example features 6 possible pairs of control actions, so there are $6 \times 8 = 48$ potential Type 3-4 UCCAs involving two actions. But the problem is actually more complicated because there can be sequences of three or even all four control actions to consider. Evaluating every potential combination becomes quickly intractable.

Table 4-1. Full Enumeration of Type 1-2 UCCAs for the Multi-UAS Example

#	c_1		c_2	
1	$\neg u_1$	$\neg u_2$	$\neg u_1$	$\neg u_2$
2	$\neg u_1$	$\neg u_2$	$\neg u_1$	u_2
3	$\neg u_1$	$\neg u_2$	u_1	$\neg u_2$
4	$\neg u_1$	$\neg u_2$	u_1	u_2
5	$\neg u_1$	u_2	$\neg u_1$	$\neg u_2$
6	$\neg u_1$	u_2	$\neg u_1$	u_2
7	$\neg u_1$	u_2	u_1	$\neg u_2$
8	$\neg u_1$	u_2	u_1	u_2
9	u_1	$\neg u_2$	$\neg u_1$	$\neg u_2$
10	u_1	$\neg u_2$	$\neg u_1$	u_2
11	u_1	$\neg u_2$	u_1	$\neg u_2$
12	u_1	$\neg u_2$	u_1	u_2
13	u_1	u_2	$\neg u_1$	$\neg u_2$
14	u_1	u_2	$\neg u_1$	u_2
15	u_1	u_2	u_1	$\neg u_2$
16	u_1	u_2	u_1	u_2

#	UAS_1		UAS_2		Context
1	¬jam	¬strike	¬jam	¬strike	*when…*
2	¬jam	¬strike	¬jam	strike	*when…*
3	¬jam	¬strike	jam	¬strike	*when…*
4	¬jam	¬strike	jam	strike	*when…*
5	¬jam	strike	¬jam	¬strike	*when…*
6	¬jam	strike	¬jam	strike	*when…*
7	¬jam	strike	jam	¬strike	*when…*
8	¬jam	strike	jam	strike	*when…*
9	jam	¬strike	¬jam	¬strike	*when…*
10	jam	¬strike	¬jam	strike	*when…*
11	jam	¬strike	jam	strike	*when…*
12	jam	¬strike	jam	strike	*when…*
13	jam	strike	¬jam	¬strike	*when…*
14	jam	strike	¬jam	strike	*when…*
15	jam	strike	jam	¬strike	*when…*
16	jam	strike	jam	strike	*when…*

Figure 4-11 illustrates the general problem with a team of n controllers that can issue m types of control actions to a shared process. The controllers may overlap in their ability to provide the same type of command for any number of these control actions. For generality, the figure shows the system having full overlap for all control actions.

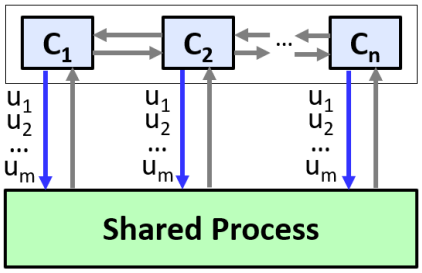


Figure 4-11. General Team of Multiple Controllers Issuing Multiple Control Actions

This team can provide up to p total control actions to the shared process, where p is defined by Equation (1). Here, $U_a(c_i) = 1$ if controller c_i can provide control action u_a, and $\boldsymbol{N}$ and $\boldsymbol{M}$ are the domains of all control actions and controllers respectively.

$$p = \sum U_i(c_a);\ \forall i\ \in \boldsymbol{N},\ \forall a \in \boldsymbol{M} \tag{1}$$

In the general problem, there are $d^{T12} = 2^p$ possible Type 1-2 UCCA combinations of n controllers *providing* or *not providing* each of the m control actions. This number grows exponentially with n and m. The number of Type 3-4 UCCA permutations, or ordered sequences, in which up to any of these p control actions can be *started* and *ended* relative to one another grows even faster, as found in Equation (2).

$$d^{T34} = \sum_{k=2}^{p} \frac{2^k p!}{(p-k)!} \tag{2}$$

Table 4-2 illustrates the total number of UCCAs for different hypothetical teams. For simplification, here every controller on the team can provide every control action, so $p = n \times m$.

Table 4-2. Number of UCCAs Enumerable for Different Hypothetical Teams

Controllers n	Types of Control Actions m	Total Control Actions p	Type 1-2 UCCAs d^{T12}	Type 3-4 UCCAs d^{T34}	Total UCCAs $d^{T12} + d^{T34}$
2	1	2	4	8	12
2	2	4	16	624	640
2	3	6	64	75,960	76,024
2	4	8	256	17,017,952	17,018,208
3	1	3	8	72	80
3	2	6	64	75,960	76,024
3	3	9	512	3.06×10^8	3.06×10^8
3	4	12	4096	3.23×10^{12}	3.23×10^{12}

As shown, teams of even modest sizes produce too many potential UCCAs to enumerate fully. It is not practical for a human analyst to identify the context(s) in which all these UCCAs are unsafe, and then develop causal scenarios to explain how they could occur. Such an effort would also be inefficient as it would produce similar information repeated across multiple similar UCCAs. Finally, the volume of data would be too overwhelming for designers to derive useful engineering and assurance decisions. Simplification is necessary.

Process of Abstraction to Manage Combinatorial Complexity

The ability to use abstraction to manage complexity is one of the key strengths of system-theoretic approaches. As shown below, collaborative systems can be systematically abstracted to reduce the combinatorial complexity associated with enumerating UCCAs. The approach, summarized in Figure 4-12, helps identify the context in which multiple control actions are unsafe together at a higher level of abstraction, where fewer combinations exist. Then, only those combinations found to be unsafe need to be further explored through refinement.

The approach begins with two different model representations, Abstractions 1a and 1b, which are shown one level up from the problem formulation in Figure 4-12. Each one focuses on a

different way to combine multiple control actions. The following discussion explains each of these abstractions and demonstrates how they enumerate UCCAs in the Multi-UAS example above. This is a necessary step to then abstract the model further to Abstraction 2a and 2b to handle more complex systems, as addressed later in Section 4.2.3.

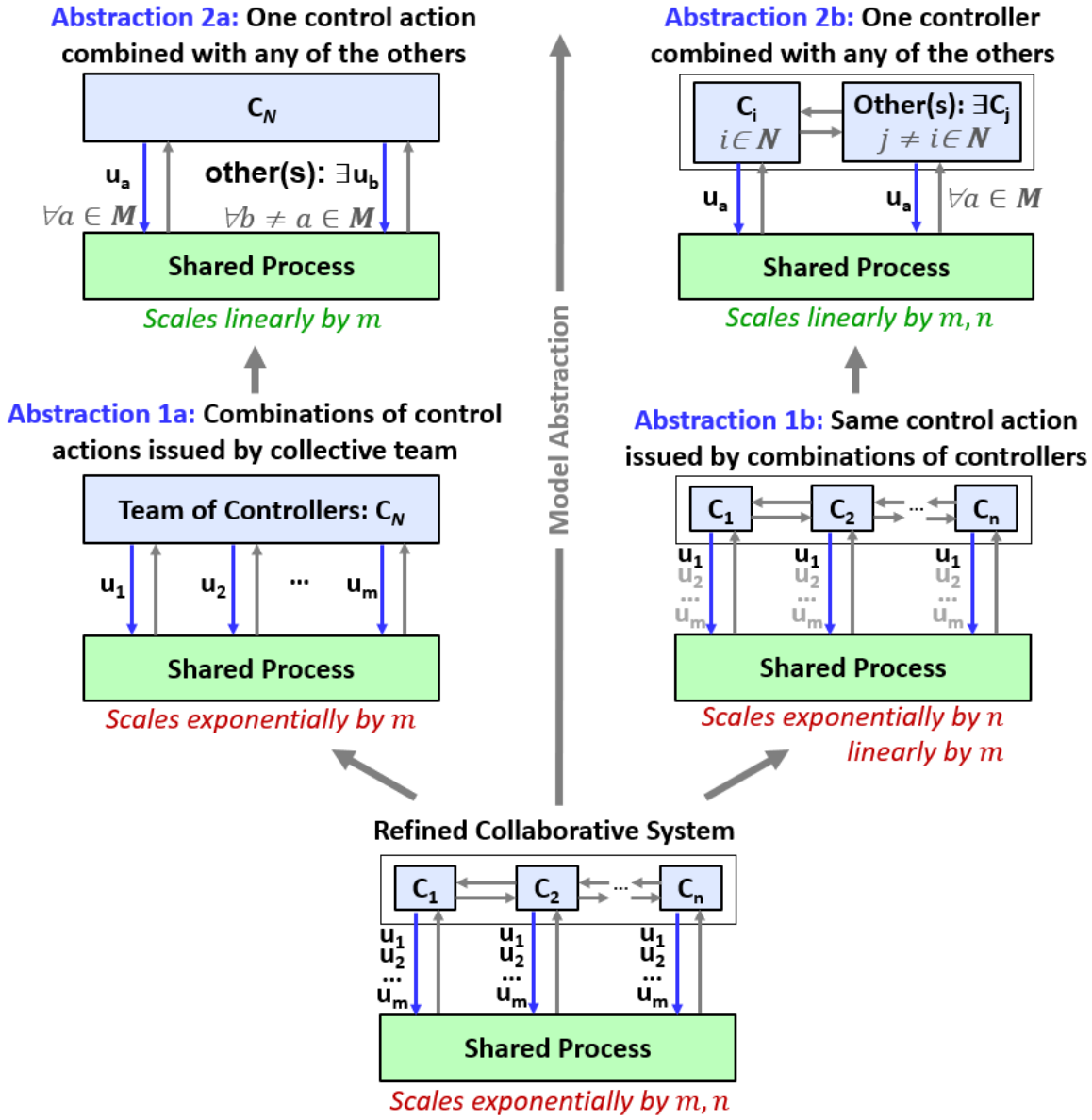


Figure 4-12. Managing Combinatorial Complexity Using Abstraction

UCCAs in Abstractions 1a: Combinations of Control Actions by Collective Team

Abstraction 1a combines the multiple controllers that share authority over a process into one collective controller (see Figure 4-12). This helps to identify the unsafe combinations of different types of control actions issued by the overall team to the process.

The use of this abstraction is predicated on several conditions. The system must exhibit shared authority between multiple controllers over a common process, or similarly, over different

interdependent subprocesses. It is also only useful if there are multiple different types of control actions to consider, or in other words for a set of control actions ($\{u_1, \ldots, u_m\} \mid m > 1$). However, it does not require the controllers to have any overlap in the types of control actions they provide.

For example, if in the multi-UAS system in Figure 4-10, only UAS_1 can *jam*, and only UAS_2 can *strike*, Abstraction 1a can be applied to explore combinations of these two commands issued by the collective team. Conversely, if the two UAS can only *jam*, and no UAS can *strike*, then this abstraction is not useful. Although in this case, Abstraction 1b is, as will be described later.

Table 4-3 shows the Type 1-2 UCCAs generated using Abstraction 1a for the multi-UAS example as defined in Figure 4-10. The table shows the combinations of control actions provided or not provided by the collective controller team, c_N.

Table 4-3. Abstraction 1a Type 1-2 UCCAs for Multi-UAS Example

#	c_N	
1	$\neg u_1$	$\neg u_2$
2	$\neg u_1$	u_2
3	u_1	$\neg u_2$
4	u_1	u_2

#	UAS Team		Context
1	¬jam	¬strike	*when…*
2	¬jam	strike	*when…*
3	jam	¬strike	*when…*
4	jam	strike	*when…*

Table 4-4 lists the Abstraction 1a Type 3-4 UCCAs, or sequences in which the collective team can start and end its control actions. In the table, $S(u)$ means "start control action u", $E(u)$ means "end u", and **F** is the Linear Temporal Logic operator for *Some **F**uture Step* [207]. The table has two columns headed by controllers, and it is read as the first controller starts/ends the control action in any row before the second controller listed starts/ends its control action in the same row. In this case, the controller is the collective team, c_N, so it is the same in both columns.

Table 4-4. Abstraction 1a Type 3-4 UCCAs for Multi-UAS Example

#	c_N	**F** c_N
1	$S(u_1)$	$S(u_2)$
2	$S(u_1)$	$E(u_2)$
3	$E(u_1)$	$S(u_2)$
4	$E(u_1)$	$E(u_2)$
5	$S(u_2)$	$S(u_1)$
6	$S(u_2)$	$E(u_1)$
7	$E(u_2)$	$S(u_1)$
8	$E(u_2)$	$E(u_1)$

#	Team	before Team	Context
1	*starts* jam	*starts* strike	*when…*
2	*starts* jam	*ends* strike	*when…*
3	*ends* jam	*starts* strike	*when…*
4	*ends* jam	*ends* strike	*when…*
5	*starts* strike	*starts* jam	*when…*
6	*starts* strike	*ends* jam	*when…*
7	*ends* strike	*starts* jam	*when…*
8	*ends* strike	*ends* jam	*when…*

These tables can be generated using automation. The human analyst then evaluates each item and determines if there are any context(s) in which that combination can lead to hazard(s). For instance, based on assumptions in the example, item 2 in Table 4-3 can be written as:

> **UCCA 1**: UAS Team **does not provide** jam and **provides** strike *when an enemy radar is surveilling the target area [H1]*.

Similarly, item 3 in Table 4-4 forms the following UCCA:

UCCA 2: UAS Team **ends providing** jam before it **starts providing** strike *when an enemy radar is surveilling the target area [H1]*.

In each case, the abstracted team can later be refined, again using automation, to explore how different combinations of controllers can issue control actions that lead to that collective UCCA output. The context and hazard traceability generated at the higher level are carried over to the refined UCCA. For example, UCCA 1 can be refined as:

UCCA 1.1: UAS_1 **does not provide** {jam or strike}; UAS_2 **does not provide** jam and **provides** strike *when an enemy radar is surveilling target area [H1]*.

UCCA 1.2: UAS_1 **does not provide** jam and **provides** strike; UAS_2 **does not provide** {jam or strike} *when an enemy radar is surveilling target area [H1]*.

UCCA 1.3: UAS_1 **does not provide** jam and **provides** strike; UAS_2 **does not provide** jam and **provides** strike *when an enemy radar is surveilling target area [H1]*.

The same process of refinement can be applied to UCCA 2. The reason for refining the UCCA is that the causal factors later analyzed by developing loss scenarios may be different for the different controllers involved. The refinement of UCCAs is further explained and formalized in Section 4.2.4.

UCCAs in Abstractions 1b: Combinations of Controllers Issuing Shared Control Action

Abstraction 1b helps to determine the unsafe combinations of controllers issuing a common control action. As reflected in Figure 4-12, it represents the multiple controllers and focuses only on one common type of control action at a time.

As was the case in Abstraction 1a, the application of 1b is also predicated on the *shared authority* of multiple controllers over a common process. There must also be some overlap between at least two of the controllers in their ability to issue a common type of control action. This overlap may be enabled by *dynamic authority* or *transfer of authority* over that control action, but that is not always the case. For example, multiple aircraft flying in formation have a standing shared control authority over aircraft separation, which can be analyzed using this abstraction.

The abstraction does not require all controllers to have common authority over all tasks. In the Multi-UAS example (Figure 4-10), if only UAS_1 can *strike*, but both UAS can *jam*, then Abstraction 1b can be applied to analyze the combination of multiple controllers issuing the *jam* command. Conversely, if only UAS_1 can *strike* and only UAS_2 can *jam*, then this abstraction is not relevant. Although in this case, it should still be analyzed with respect to Abstraction 1a as previously described.

Table 4-5 and Table 4-6 enumerate Type 1-2 and Type 3-4 combinations of controllers issuing the same control action using Abstraction 1b. The tables focus on control action u_1, the *jam* command in the multi-UAS example, and would be duplicated for u_2, *strike*. Here, $U(x)$ means "control action u is provided by controller x", where x is enumerated as c_1 and c_2. All other conventions are consistent with those previously defined.

Table 4-5. Abstraction 1b Type 1-2 UCCAs for Multi-UAS Example

#	$U_1(x)$		Common Issues
1	$\neg U_1(c_1)$	$\neg U_1(c_2)$	control gap
2	$\neg U_1(c_1)$	$U_1(c_2)$	controller-task mismatch
3	$U_1(c_1)$	$\neg U_1(c_2)$	
4	$U_1(c_1)$	$U_1(c_2)$	control overlap

#	jam provided by		Context
1	¬UAS$_1$	¬UAS$_2$	*when…*
2	¬UAS$_1$	UAS$_2$	*when…*
3	UAS$_1$	¬UAS$_2$	*when…*
4	UAS$_1$	UAS$_2$	*when…*

Table 4-6. Abstraction 1b Type 3-4 UCCAs for Multi-UAS Example

#	$U_1(x)$	**F** $U_1(x)$	Common Issues
1	$S(U_1(c_1))$	$S(U_1(c_2))$	
2	$S(U_1(c_1))$	$E(U_1(c_2))$	handoff overlap
3	$E(U_1(c_1))$	$S(U_1(c_2))$	handoff gap
4	$E(U_1(c_1))$	$E(U_1(c_2))$	
5	$S(U_1(c_2))$	$S(U_1(c_1))$	
6	$S(U_1(c_2))$	$E(U_1(c_1))$	handoff overlap
7	$E(U_1(c_2))$	$S(U_1(c_1))$	handoff gap
8	$E(U_1(c_2))$	$E(U_1(c_1))$	

#	jam by	before jam by	Context
1	UAS$_1$ *starts*	UAS$_2$ *starts*	*when…*
2	UAS$_1$ *starts*	UAS$_2$ *ends*	*when…*
3	UAS$_1$ *ends*	UAS$_2$ *starts*	*when…*
4	UAS$_1$ *ends*	UAS$_2$ *ends*	*when…*
5	UAS$_2$ *starts*	UAS$_1$ *starts*	*when…*
6	UAS$_2$ *starts*	UAS$_1$ *ends*	*when…*
7	UAS$_2$ *ends*	UAS$_1$ *starts*	*when…*
8	UAS$_2$ *ends*	UAS$_1$ *ends*	*when…*

The systematic consideration of how different controllers provide, don't provide, start, and end a particular command highlights potential gaps, overlaps, mismatches, and handoff issues that are common in shared control. For example, item 1 in Table 4-5, which involves no controller providing the command, may be due to a gap in task allocation associated with *dynamic authority*. Items 2 and 3 explore how the "wrong" controller provides an action as can occur in a controller-task mismatch. Item 4 considers multiple controllers providing the action, which may lead to control conflicts.

Table 4-6 highlights related issues. In items 2 and 6, one controller starts providing the command before another controller ends it. This may occur in a *transfer of authority* when the handoff involves an excessive period of overlap between the two controllers, which can result in conflicting control. Items 3 and 7 reflect the opposite problem when one controller ends before the next starts. This can lead to a period of gap in authority when the process is not controlled. The overlap and gap in control are visible in items 2 and 3, respectively, in Figure 4-9.

As mentioned for Abstraction 1a, the tables can be generated using automation, and an analyst can then specify if there are any context(s) and hazard(s) that apply to each item. For example, item 4 in Table 4-5 can generate the following UCCA. In this case, there is no need to refine the UCCA as it already provides the controllers and control actions involved.

> **UCCA 3**: UAS$_1$ **provides** jam and UAS$_2$ **provides** jam *when both controllers interfere with each other [H2]*.

Summary of Abstractions 1a and 1b

By applying Abstractions 1a and 1b to the multi-UAS example, the number of potential UCCAs is reduced from 640 in the fully enumerated set, as listed in Table 4-2 for $n = 2$ and $m = 2$, down to 36. There are now 12 Type 1-2 UCCAs and 24 Type 3-4 UCCAs for a human to analyze.

Unfortunately, while the number of UCCAs grows more slowly than in the fully refined problem, at this level of abstraction, it still scales exponentially. Abstraction 1a is, by definition, combinatorial in the number of different types of control actions m. Abstraction 1b grows exponentially with the number of controllers n. And in both cases, the permutations of Type 3-4 UCCAs scale even faster once they involve more than the simple pairwise comparisons illustrated so far. Further simplification is necessary.

4.2.3 Linearizing Growth by Abstracting Further

The method of abstraction applied in the previous section provides the first step to mitigate exponential scaling in the number of UCCAs. This section introduces an additional level of abstraction, shown at the top of Figure 4-12, which linearizes the growth of potential UCCAs for collaborative systems of any number of controllers and control actions.

The previous multi-UAS example is slightly modified to illustrate the process. Consider now a team of three unmanned systems (UxS), consisting of 2 UAS and 1 ground robot ($n = 3$). Each controller can execute the same types of control actions: *jam*, *strike*, and *track* a target ($m = 3$). Figure 4-13 shows the updated control structure for this concept alongside its generalized form.

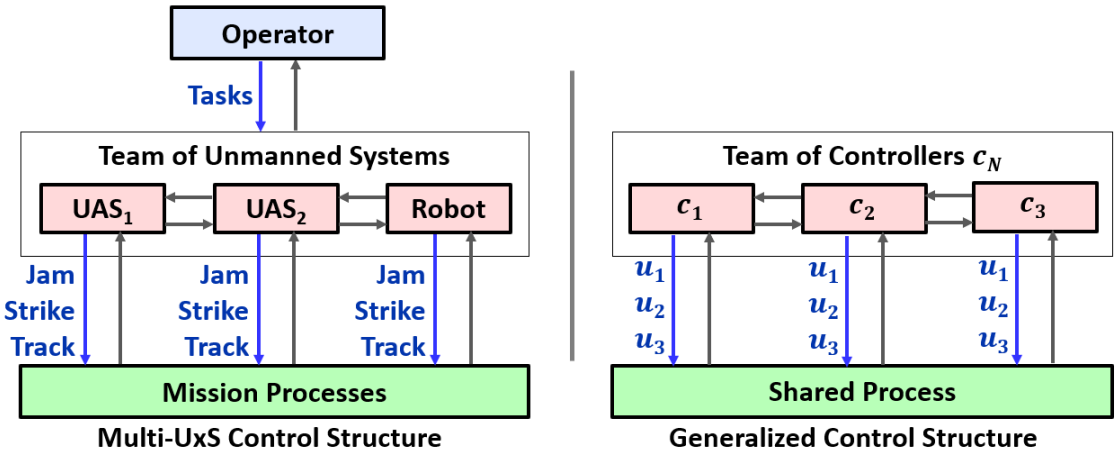


Figure 4-13. Updated Multi-UxS Control Structure (left) and its Generalized Form (right)

To reduce the number of potential UCCAs, the system is first represented using Abstractions 1a and 1b as described in the previous section. Each model is then further abstracted, respectively, to Abstractions 2a and 2b, as shown in Figure 4-12.

Figure 4-14 shows this process for the multi-UxS example. The figure only depicts one iteration of Abstractions 2a and 2b centered on control action u_1. It also explains how to reiterate the cases for u_2 and u_3. Finally, the figure clarifies that a system can be directly represented by Abstractions 2a and 2b. Abstractions 1a and 1b simply provide the system-theoretic foundation for the higher-level representation, but no analysis is necessary at that level.

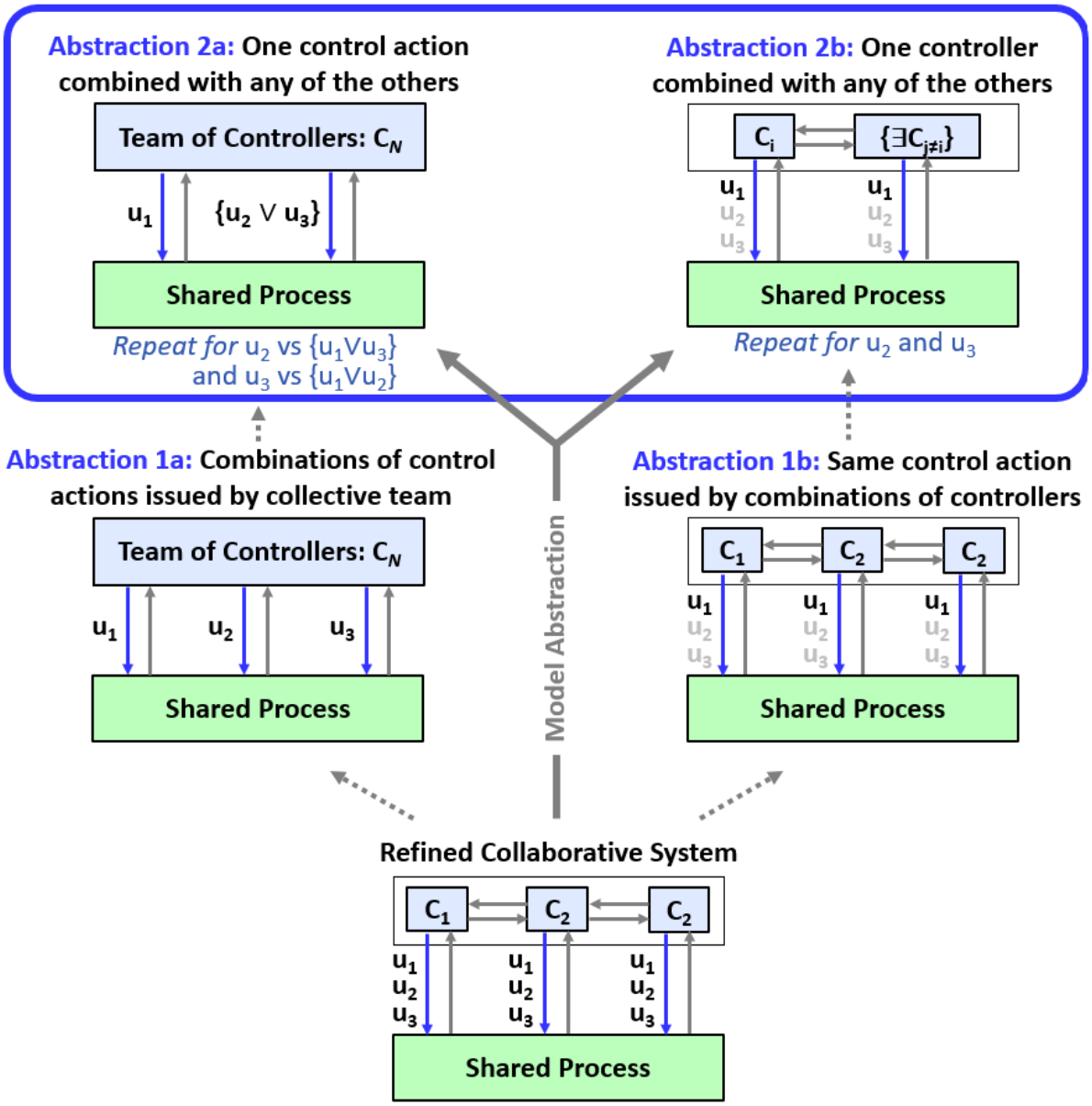


Figure 4-14. Abstraction of a Collaborative System to Models 2a and 2b

UCCAs in Abstraction 2a

Abstraction 2a is based on Abstraction 1a, and it iteratively considers any one of the control actions in combination with any of the others abstracted together (see Figure 4-14). At this level, Type 1-2 UCCAs explore how the collective team provides (or does not provide) any one control action and provides (or does not provide) any of the others. One iteration of this process comparing u_1 with $\{u_2 \vee u_3\}$ for the modified multi-UxS example is:

1. c_N **does not provide** u_1 and **does not provide** $\{u_2$ or $u_3\}$ when… *[H]*
2. c_N **does not provide** u_1 and **provides** $\{u_2$ or $u_3\}$ when… *[H]*
3. c_N **provides** u_1 and **does not provide** $\{u_2$ or $u_3\}$ when… *[H]*
4. c_N **provides** u_1 and **provides** $\{u_2$ or $u_3\}$ when… *[H]*

In these statements, $\{u_2$ or $u_3\}$ is true under the following conditions: u_2 alone is true, u_3 alone is true, or u_2 and u_3 are both true. Two additional iterations compare u_2 with $\{u_1 \vee u_3\}$ and u_3 with $\{u_1 \vee u_2\}$. Their formulations are comparable to those listed above. The full set of potential Type 1-2 UCCAs for Abstraction 2a of the multi-UxS system is shown in Table 4-7.

Table 4-7. Abstraction 2a Type 1-2 UCCAs for Multi-UxS Example

#	c_N	
1	$\neg u_1$	$\neg\{u_2 \vee u_3\}$
2	$\neg u_1$	$\{u_2 \vee u_3\}$
3	u_1	$\neg\{u_2 \vee u_3\}$
4	u_1	$\{u_2 \vee u_3\}$
5	$\neg u_2$	$\neg\{u_1 \vee u_3\}$
6	$\neg u_2$	$\{u_1 \vee u_3\}$
7	u_2	$\neg\{u_1 \vee u_3\}$
8	u_2	$\{u_1 \vee u_3\}$
9	$\neg u_3$	$\neg\{u_1 \vee u_2\}$
10	$\neg u_3$	$\{u_1 \vee u_2\}$
11	u_3	$\neg\{u_1 \vee u_2\}$
12	u_3	$\{u_1 \vee u_2\}$

#	UxS Team		Context
1	¬jam	¬any in{strike, track}	*when…*
2	¬jam	any in{strike, track}	*when…*
3	jam	¬any in{strike, track}	*when…*
4	jam	any in{strike, track}	*when…*
5	¬strike	¬any in{jam, track}	*when…*
6	¬strike	any in{jam, track}	*when…*
7	strike	¬any in{jam, track}	*when…*
8	strike	any in{jam, track}	*when…*
9	¬track	¬any in{jam, strike}	*when…*
10	¬track	any in{jam, strike}	*when…*
11	track	¬any in{jam, strike}	*when…*
12	track	any in{jam, strike}	*when…*

Type 3-4 UCCAs follow a similar concept. They represent ways in which the collective team may start or end any one control action before starting or ending any of the others. One of three iterations of Type 3-4 UCCAs in Abstraction 2a is listed below and the full set for the multi-UxS example is provided in Table 4-8.

1. c_N **starts providing** u_1 before it **starts providing** $\{u_2$ or $u_3\}$ when… *[H]*
2. c_N **starts providing** u_1 before it **ends providing** $\{u_2$ or $u_3\}$ when… *[H]*
3. c_N **ends providing** u_1 before it **starts providing** $\{u_2$ or $u_3\}$ when… *[H]*
4. c_N **ends providing** u_1 before it **ends providing** $\{u_2$ or $u_3\}$ when… *[H]*

In the statements above, if either u_2 or u_3 is a discrete command, it must be removed from consideration in specifying the UCCA contexts that involve "ends providing $\{u_2$ or $u_3\}$". This removal can be automated. For example, if u_2 is a discrete command and u_3 is a continuous command, then the second line must be interpreted as:

2. c_N **starts providing** u_1 before it **ends providing** u_3 when… *[H]*

Similarly, if both u_2 and u_3 are discrete commands, the statement is removed altogether. However, statements 1 and 3, which involve starting these discrete commands, would still be valid to consider.

Table 4-8. Abstraction 2a Type 3-4 UCCAs for Multi-UxS Example

#	c_N	**F** c_N
1	$S(u_1)$	$S(\{u_2 \vee u_3\})$
2	$S(u_1)$	$E(\{u_2 \vee u_3\})$
3	$E(u_1)$	$S(\{u_2 \vee u_3\})$
4	$E(u_1)$	$E(\{u_2 \vee u_3\})$
5	$S(u_2)$	$S(\{u_1 \vee u_3\})$
6	$S(u_2)$	$E(\{u_1 \vee u_3\})$
7	$E(u_2)$	$S(\{u_1 \vee u_3\})$
8	$E(u_2)$	$E(\{u_1 \vee u_3\})$
9	$S(u_3)$	$S(\{u_1 \vee u_2\})$
10	$S(u_3)$	$E(\{u_1 \vee u_2\})$
11	$E(u_3)$	$S(\{u_1 \vee u_2\})$
12	$E(u_3)$	$E(\{u_1 \vee u_2\})$
13*	$S(\{u_2 \vee u_3\})$	$S(u_1)$
…	…	…
24*	$E(\{u_1 \vee u_2\})$	$E(u_3)$

#	Team	before Team	Context
1	*starts* jam	*starts* any in{strike, track}	*when…*
2	*starts* jam	*ends* any in{strike, track}	*when…*
3	*ends* jam	*starts* any in{strike, track}	*when…*
4	*ends* jam	*ends* any in{strike, track}	*when…*
5	*starts* strike	*starts* any in{jam, track}	*when…*
6	*starts* strike	*ends* any in{jam, track}	*when…*
7	*ends* strike	*starts* any in{jam, track}	*when…*
8	*ends* strike	*ends* any in{jam, track}	*when…*
9	*starts* track	*starts* any in{jam, strike}	*when…*
10	*starts* track	*ends* any in{jam, strike}	*when…*
11	*ends* track	*starts* any in{jam, strike}	*when…*
12	*ends* track	*ends* any in{jam, strike}	*when…*
13*	*starts* any in{strike, track}	*starts* jam	*when…*
…	…	…	…
24*	*ends* any in{jam, strike}	*ends* track	*when…*

**Items 13-24 are the reverse sequences of Items 1-12 respectively*

As in Abstractions 1a and 1b, these tables can also be generated by automation, and the same approach is employed to identify and refine UCCAs (see Section 4.2.2). However, one notable difference is that in Abstraction 2a the UCCA must also specify what control actions in the abstracted set are relevant to the context. This determination must be made by the human analyst because the context is defined by the analyst. For example, item 10 in Table 4-7 may produce:

> **UCCA 4**: UAS Team **does not provide** track and **provides** strike *when an enemy must be tracked as a strike occurs to ensure custody of the target [H3]*.

This UCCA is uncovered by analyzing the ¬*track* and {*jam* ∨ *strike*} combination of commands. However, in the set {*jam* ∨ *strike*}, only the *strike* command is relevant. Under the context specified, it does not matter if *jam* is provided or not. By designating the relevant commands, the refined UCCA only includes the different controller options that lead to collectively not tracking and striking (see Table 4-9). The exclusion of *jam* eliminates additional combinations that are invariant to the context and subsequent causal analysis of the UCCA.

Table 4-9. Refinement of Example UCCA 4

UCCA	Team	Team	Context
4	¬track	{~~jam~~ ∨ strike}	*when enemy must be tracked as strike occurs [H3]*

Refined	UAS_1	UAS_2	robot	Same Context and Hazard
4.1	¬track	¬track	strike	*when enemy must be tracked as strike occurs [H3]*
4.2	¬track	strike	¬track	*when enemy must be tracked as strike occurs [H3]*
4.3	strike	¬track	¬track	*when enemy must be tracked as strike occurs [H3]*
4.4	¬track	strike	strike	*when enemy must be tracked as strike occurs [H3]*
4.5	strike	¬track	strike	*when enemy must be tracked as strike occurs [H3]*
4.6	strike	strike	¬track	*when enemy must be tracked as strike occurs [H3]*
4.7	strike	strike	strike	*when enemy must be tracked as strike occurs [H3]*

In other UCCAs, the analyst may choose to include all of the control actions listed in the combination. For example, a version of the previous UCCA 1 updated for this modified UxS example may form UCCA 5 below.

> **UCCA 5**: UAS Team **does not provide** jam and **provides** track and strike *when an enemy radar is surveilling the target area [H1]*.

UCCAs in Abstraction 2b

Abstraction 2b follows a similar logic as in 2a and uses Abstraction 1b as a foundation to consider how any one controller in combination with any of the others abstracted together may issue a common control action (see Figure 4-14). Here, Type 1-2 UCCAs explore ways the one controller and the others provide or do not provide the action, as shown in Table 4-10. The process is reiterated for each control action.

Table 4-10. Abstraction 2b Type 1-2 UCCAs for Multi-UxS Example

#	$U_1(x)$		Common Issues
1	$\neg U_1(c_i)$	$\neg U_1(\{c_{j1} \vee c_{j2}\})$	gap
*	$\neg U_1(c_i)$	$U_1(\{c_{j1} \vee c_{j2}\})$	
2	$U_1(c_i)$	$\neg U_1(\{c_{j1} \vee c_{j2}\})$	mismatch
3	$U_1(c_i)$	$U_1(\{c_{j1} \vee c_{j2}\})$	overlap

#	jam provided by		Context
1	¬any one	¬any of others	*when…*
*	¬any one	any of others	*when…*
2	any one	¬any of others	*when…*
3	any one	any of others	*when…*

**Does not need to be considered, included in items 2 and 3 (see discussion)*

A key difference in how UCCAs are defined in Abstraction 2b, in this work, is that the specific controllers do not need to be listed in the abstracted form. For the systems explored in this research, it is assumed that the context in which a control action, or a combination of control actions, is unsafe is controller agnostic. As such, the controllers listed in the abstracted UCCA (e.g., those in Table 4-10) are generalized, which reduces the number of combinations that need to be analyzed at this level.

However, it is important to note that the specific controllers must be considered in the refinement of the UCCA. This is a necessary step to explore how those different controllers can contribute to that higher-level output. By listing these options, the causal factors that are controller-specific can later be identified in the last step of the hazard analysis, scenario development. These factors would otherwise be missed. The process of refining a UCCA can be fully automated, and the context and hazard identified by the human analyst at the higher level are carried over to the refined level, as shown below.

An example of a UCCA found in the multi-UxS system using Abstraction 2b follows. The last combination in Table 4-10, in which multiple controllers provided the *jam* command, is refined in Table 4-11.

> **UCCA 6**: any one C_i **provides** jam and any other C_j **provide(s)** jam *when multiple controllers interfere with each other [H2]*.

Table 4-11. Refinement of Example UCCA 5

UCCA	C_i	Any other C_j		Context
6	jam	jam		*when multiple controllers mutually interfere [H2]*
Refined	UAS_1	UAS_2	robot	Same Context and Hazard
6.1		jam	jam	*when multiple controllers mutually interfere [H2]*
6.2	jam		jam	*when multiple controllers mutually interfere [H2]*
6.3	jam	jam		*when multiple controllers mutually interfere [H2]*
6.4	jam	jam	jam	*when multiple controllers mutually interfere [H2]*

The refinement in Table 4-11 assumes all controllers can issue all control actions. If instead, only UAS_1 and UAS_2 could provide the *jam* command, and not the robot, then item 6.3 would be the only option in the refinement.

Another advantage to generalizing the controllers in the UCCA identification is that the second row labeled (*) in Table 4-10 can be skipped. This line represents one controller c_i not providing the command when any of the others in $\{c_{j\neq i}\}$ provide it. If only one controller in $\{c_{j\neq i}\}$ provides the command, that is addressed by item 2 of the table. If multiple controllers in $\{c_{j\neq i}\}$ provide it, that is considered in item 3.

Despite the benefits, there may be circumstances in which an engineering team chooses not to make the assumption that context is controller agnostic. In such cases, the same mechanism used to enumerate multiple different control actions in Abstraction 2a can be applied to enumerate multiple specific controllers. All four items per iteration in Table 4-10 would need to be considered, the process would be reiterated to compare each controller to the others, and the analyst could specify which controllers are relevant to the contexts identified. The reasons for selecting this approach are beyond the scope of this dissertation.

Type 3-4 UCCAs in this abstraction explore the sequences in which any one controller and any other start or end a common control action. Table 4-12 shows the process for one control action (u_1), and similar iterations are necessary for each of the other control actions.

By again assuming that the context is controller agnostic, the process only needs to consider any one of the other controllers in $\{c_{j\neq i}\}$ at a time. Similar to above, if an engineering team does not want to make this assumption, it can enumerate options such as c_i starts/ends u_1 before any controllers in $\{c_{j1}, \ldots, c_{j(n-1)}\}$ start/end u_1. The enumeration would also need to include the reverse sequences, and the analyst would specify the controllers relevant to the context identified.

Table 4-12. Abstraction 2b Type 3-4 UCCAs for Multi-UxS Example

#	$U_1(x)$	**F** $U_1(x)$	Common Issues
1	$S(U_1(c_i))$	$S(U_1(c_j))$	
2	$S(U_1(c_i))$	$E(U_1(c_j))$	handoff overlap
3	$E(U_1(c_i))$	$S(U_1(c_j))$	handoff gap
4	$E(U_1(c_i))$	$E(U_1(c_j))$	

#	jam by	before jam by	Context
1	any one *starts*	any other *starts*	*when…*
2	any one *starts*	any other *ends*	*when…*
3	any one *ends*	any other *starts*	*when…*
4	any one *ends*	any other *ends*	*when…*

As discussed for Abstraction 1b, Abstraction 2b also highlights the same common issues with shared control that may contribute to entering a hazardous state. The ability to systematically identify potential control gaps, overlaps, mismatches, and unsafe handoffs as listed in Table 4-10 and Table 4-12 is a key strength of the overall UCCA approach.

Assumptions and Limitations

The abstractions employed to enumerate UCCAs presented in this section provide tractability to an otherwise combinatorial problem. However, the method involves certain assumptions and has some limitations that are important to understand.

The following are the key assumptions. First, the abstractions are predicated on multiple controllers sharing authority over a common process or over different interdependent subprocesses. Second, Abstractions 1a and 2a assume these controllers, collectively, can provide multiple types of control actions to the shared process. Third, Abstractions 1b and 2b assume multiple controllers can provide the same type of control action to the process. Fourth, in this work, the context of the UCCA is assumed to be agnostic to the controllers that provide the actions. Mechanisms to relax this assumption are addressed in the discussion above. Finally, it is assumed a human analyst is able to identify the unsafe context of a UCCA given the abstracted combination of control actions or controllers.

The main objective of the abstractions is to reduce the number of potential UCCAs to consider in arbitrarily complex systems. However, in the multi-UxS example (Figure 4-13), the process generates more Type 1-2 UCCAs (provide / not provide) in Abstraction 2a than in Abstraction 1a. In Abstraction 2a, the UCCA count grows linearly by $4m$, instead of exponentially by 2^m in 1a, where m is the number of different types of control actions (here, $m = 3$).

The numerical advantage of the additional abstraction is realized when $m > 4$. This means an argument can be made to enumerate Type 1-2 UCCAs using Abstraction 1a instead of 2a for systems with 4 or fewer control actions. Anecdotally, the author found it easier cognitively to explore combinations of more than 3 control actions using Abstraction 2a. Similarly, because controllers are defined generally, Abstraction 2b is advantageous over Abstraction 1b when there are three or more controllers.

Abstractions 2a and 2b provide even greater benefits for Type 3-4 UCCAs (start/end before others start/end). The abstraction leverages some of the numerical advantages of pairwise analysis by avoiding permutations of sequences greater than two. However, by maintaining consideration for all controllers or control actions at a time, the abstraction retains a broader analytical scope than if the problem was reduced to only evaluate pairs. Such a reduction could never consider interactions that involve more than two controllers or control actions.

One limitation is that some control combinations may be missed due to the method of abstraction. Table 4-13 provides representative examples of UCCAs that are omitted from the abstracted enumeration and an explanation for why they are not covered.

In all cases, simpler combinations that address a subset of the overall interaction are identified. However, as informed by Systems Theory, the aggregate of these reduced cases may not necessarily represent the behavior of the full interaction. Still, these cases may help analysts consider the more complex combinations that are missed by the abstraction.

Table 4-13. Examples of Control Combinations Not Addressed Using Full Abstraction

Missed UCCA	Reason it is Missed
C_N **provides** u_1 and u_2 and **does not provide** u_3 and u_4 $u_1 \wedge u_2 \wedge \neg u_3 \wedge \neg u_4$	Abstraction 2a considers whether 1 control action is provided (or not), and if any of the others are provided (or not). However, this UCCA is considered in Abstraction 1a. Simpler combinations addressed in Abstraction 2a: $(u_1 \wedge u_2 \wedge \neg u_3)$; $(u_1 \wedge u_2 \wedge \neg u_4)$; $(u_1 \wedge \neg u_3 \wedge \neg u_4)$; $(u_2 \wedge \neg u_3 \wedge \neg u_4)$
C_i **ends** u_1 before (C_j **starts** u_2 and C_j **ends** u_3)	Similar to above, the abstraction applies Start or End to the other commands together. Simpler combinations addressed: C_i **ends** u_1 before C_j **starts** u_2; C_i **ends** u_1 before C_j **ends** u_3
C_i **ends** u_1 before (C_j **starts** u_1 and C_i **starts** u_2)	The combination bridges over the initial decision to look at combinations of different control actions, and combinations of controllers issuing the same control action. Simpler combinations addressed: C_i **ends** u_1 before C_j **starts** u_1; C_i **ends** u_1 before C_i **starts** u_2
C_i **ends** u_1 before C_j **starts** u_2 before C_i **starts** u_3	This is a sequence of three events, which is beyond what is considered. Simpler combinations addressed: C_i **ends** u_1 before {C_j **starts** u_2 and C_i **starts** u_3}**;** C_j **starts** u_2 before C_i **starts** u_3
C_1 **ends** u_1 before (C_2 **starts** u_1 and C_3 **starts** u_1)	Abstraction 2b Type 3-4 UCCAs only consider any one controller and any other one by assuming the context is controller agnostic (by choice only, see discussion) Simpler combinations addressed: C_1 **ends** u_1 before C_2 **starts** u_1; C_1 **ends** u_1 before C_3 **starts** u_1

To illustrate one of these missed combinations and its simpler cases considered, the second line of the table may represent the following ordered sequence for the multi-UxS example in Figure 4-13.

Missed: Any C_i **ends** jam before any others in C_j **start** strike and **ends** track *when …*

Simpler considered: Any C_i **ends** jam before any other C_j **starts** strike *when …*

Simpler considered: Any C_i **ends** jam before any other C_j **ends** track *when …*

The last item in Table 4-13 occurs because of the decision to assume the context of a UCCA is controller agnostic. Here, the missed UCCA does not need to be considered as long as the simpler combinations listed are addressed. However, as discussed in the previous subsection, the analysis team can choose not to make this assumption and recover the ability to find the missed UCCA. As such, this case represents a deliberate implementation decision rather than a limitation of the method.

Some of the other cases in the table may be found by arbitrarily extending the scope of the abstraction. For example, Type 1-2 UCCAs could have an additional step to compare any two controllers with all the others. Similarly, Type 3-4 UCCAs could consider some sequences of three changes in actions. These strategies may help if an analyst finds a pattern of combinations to consider that is just beyond the reach of the current approach.

However, if all higher-order combinations must be found, the guaranteed method to find them is to perform a full enumeration as described at the beginning of this section. Despite the limitations above, the overall approach to identifying UCCAs is shown in the Chapter 5 case study to be practical and find causal information that was previously not identified.

4.2.4 UCCA Identification Algorithm

The concepts developed in the previous subsections are now integrated into an end-to-end algorithm (Algorithm 1) to identify Unsafe Combinations of Control Actions (UCCAs). The algorithm introduces new concepts to reduce and prioritize the output set of UCCAs and improve the efficiency of scenario development.

Algorithm 1: UCCA Identification

Input: $\mathcal{A}$, $\mathcal{C}^{int}$, $\mathcal{S}$
Output: $\mathcal{U}^{abs}$, $\mathcal{U}^{ref}$ sets of UCCAs, abstracted & refined (Tuple)
// $\mathcal{A}$: what controller can provide what control action (Tuple)
// $\mathcal{C}^{int}$: set of interchangeable controllers (Set)
// $\mathcal{S}$: special interactions to consider in refinement (Tuple)

1. $\mathcal{C}^{abs} \leftarrow$ **Enumerate-Combinations** ($\mathcal{A}$) // [Table 4-14]*
2. for $x \in \mathcal{C}^{abs}$**
3. if Context($\mathcal{C}_x^{abs}$) $\neq \emptyset$**
4. $\mathcal{U}^{abs} = \mathcal{U}^{abs} \cup (\mathcal{C}_x^{abs}, context_x, \mathcal{U}_x^{rel})$**
5. for $x \in \mathcal{U}^{abs}$*
6. $\mathcal{C}_x^{ref} \leftarrow$ **Refine-Combinations** ($\mathcal{U}_x^{abs}$, $\mathcal{A}$, $\mathcal{S}$) // [Equation (17)]*
7. $\mathcal{C}_x^{ref\prime} \leftarrow$ **Prune-Equivalent**($\mathcal{C}_x^{ref}$, $\mathcal{C}^{int}$) // [Equation (21)]*
8. $\mathcal{U}_x^{ref} \leftarrow$ **Prioritize**($\mathcal{C}_x^{ref\prime}$, $\mathcal{S}$) // [Heuristic, see discussion]*
9. **return** ($\mathcal{U}^{abs}$, $\mathcal{U}^{ref}$)*

// * Step automated; ** Step performed by human

The UCCA Identification Algorithm includes portions that are automated (denoted by *) and others that are performed by a human analyst (**). A description of each component in the algorithm follows and includes reasoning for allocating that part of the process to the automation or the human. Some of the components are illustrated using the multi-UxS system example from Figure 4-13. A prototype tool that executes the automated portions of Algorithm 1 was developed in MATLAB and is employed in the case study in Chapter 5. The implementation of the tool is discussed where relevant.

Inputs

The first input to the algorithm is the authority tuple, $\boldsymbol{\mathcal{A}} = f(\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{A})$, which describes the set of control actions each controller can provide to the shared process. $\boldsymbol{\mathcal{A}}$ is derived from the control structure, and its components are:

> $\boldsymbol{N} \coloneqq$ vector of controllers that share authority over the process
>
> $\boldsymbol{M} \coloneqq$ vector of control actions that can be provided to the process
>
> $\boldsymbol{A} \coloneqq$ vector $\{A_a(c_i)\}$, $\forall a \in \boldsymbol{M}$, $\forall i \in \boldsymbol{N}$, for which elements are true if controller c_i can provide command u_a

The second input is the set of interchangeable controllers, $\boldsymbol{C^{int}}$. It is used to prune refined UCCAs that are considered equivalent to others in terms of potential causal factors. This topic is further addressed in the discussion about Line 7.

The third input is a set of special interactions, $\boldsymbol{S}$, encoded to add or remove refined UCCAs when enumerated. Special interactions can also be used to influence prioritization. They are further discussed in the descriptions of Lines 6 and 8.

Example: for the multi-UxS system in Figure 4-13, these terms are captured in Equations (3)-(7) below. $\boldsymbol{S}$ is addressed later in the discussion.

$$\boldsymbol{N} = \{UAS_1, UAS_1, robot\} \tag{3}$$

$$\boldsymbol{M} = \{jam, strike, track\} \tag{4}$$

$$\boldsymbol{A} = \{A_{jam}(UAS_1), A_{strike}(UAS_1), \dots, A_{track}(robot)\} = \{1,1,\dots,1\} \tag{5}$$

$$\boldsymbol{\mathcal{A}} = \begin{pmatrix} UAS_1 & jam & 1 \\ UAS_1 & strike & 1 \\ UAS_1 & track & 1 \\ UAS_2 & jam & 1 \\ UAS_2 & strike & 1 \\ UAS_2 & track & 1 \\ robot & jam & 1 \\ robot & strike & 1 \\ robot & track & 1 \end{pmatrix} \tag{6}$$

$$\boldsymbol{C^{int}} = \{UAS_1, UAS_2\} \tag{7}$$

Line 1: Automated Enumeration of Control Action Combinations

The first line in Algorithm 1 is a function that enumerates all the combinations of control actions by implementing the procedures introduced in Sections 4.2.1-4.2.3. It finds the Type 1-2 (provide/not provide) and the Type 3-4 (start/end before start/end) UCCAs using Abstractions 2a (combination of actions issued by the team) and 2b (combinations of controllers issuing a common action).

Table 4-14 formalizes this process. The first row outputs potential Type 1-2 UCCAs using Abstraction 2a in a format consistent with Table 4-7 for the multi-UxS example. These

combinations are provided by the collective team c_N of $\boldsymbol{N}$ controllers. The four cases are reiterated for every u_a in $\boldsymbol{M}$. For example, item (c) in that row describes the team providing u_a and not providing any of the other control actions, denoted by $\neg\exists u_b$.

The second row finds Type 1-2 UCCAs using Abstraction 2b like those shown in Table 4-8. The output keeps the controllers c_i and $\exists c_j$ general, where $\exists c_j$ denotes any the other controllers that are not c_i, and iterates for every u_a in $\boldsymbol{M}$. Here, item (c) in that row describes any one controller c_i providing u_a and any of the other controllers, $\exists c_j$, providing the same action.

Table 4-14. Formalized Method of Enumerating Combinations of Control Actions

Abstraction & UCCA Type	Cases to Consider		Enumerate Each Case
Abs 2a Type 1-2	a. $\neg u_a \wedge \neg\exists u_b$ b. $\neg u_a \wedge \exists u_b$ c. $u_a \wedge \neg\exists u_b$ d. $u_a \wedge \exists u_b$	given $U_a(c_N), U_b(c_N)$	for $\forall a \neq b \in \boldsymbol{M}$
Abs 2b Type 1-2	a. $\neg U_a(c_i) \wedge \neg\exists c_j\, U_a(c_j)$ b. $U_a(c_i) \wedge \neg\exists c_j\, U_a(c_j)$ c. $U_a(c_i) \wedge \exists c_j\, U_a(c_j)$	given $i \neq j \in \boldsymbol{N}$	for $\forall a \in \boldsymbol{M}$
Abs 2a Type 3-4	a. $\exists u_b[(\neg u_a \wedge \neg u_b)\ \mathbf{U}\ (u_a \wedge \neg u_b)\ \mathbf{F}\ u_b]$ b. $\exists u_b[(\neg u_a \wedge u_b)\ \mathbf{U}\ (u_a \wedge u_b)\ \mathbf{F}\ \neg u_b]$ c. $\exists u_b[(u_a \wedge \neg u_b)\ \mathbf{U}\ (\neg u_a \wedge \neg u_b)\ \mathbf{F}\ u_b]$ d. $\exists u_b[(u_a \wedge u_b)\ \mathbf{U}\ (\neg u_a \wedge u_b)\ \mathbf{F}\ \neg u_b]$ e. $\exists u_b[(\neg u_a \wedge \neg u_b)\ \mathbf{U}\ (\neg u_a \wedge u_b)\ \mathbf{F}\ u_a]$ f. $\exists u_b[(\neg u_a \wedge u_b)\ \mathbf{U}\ (\neg u_a \wedge \neg u_b)\mathbf{F}\ u_a]$ g. $\exists u_b[(u_a \wedge \neg u_b)\ \mathbf{U}\ (u_a \wedge u_b)\ \mathbf{F}\ \neg u_a]$ h. $\exists u_b[(u_a \wedge u_b)\ \mathbf{U}\ (u_a \wedge \neg u_b)\ \mathbf{F}\ \neg u_a]$	given $U_a(c_N), U_b(c_N)$	for $\forall a \neq b \in \boldsymbol{M}$
Abs 2b Type 3-4	a. $(\neg U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{U}\ (U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{F}\ U_a(c_j)$ b. $(\neg U_a(c_i) \wedge U_a(c_j))\ \mathbf{U}\ (U_a(c_i) \wedge U_a(c_j))\ \mathbf{F}\ \neg U_a(c_j)$ c. $(U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{U}\ (\neg U_a(c_i) \wedge \neg U_a(c_j))\ \mathbf{F}\ U_a(c_j)$ d. $(U_a(c_i) \wedge U_a(c_j))\ \mathbf{U}\ (\neg U_a(c_i) \wedge U_a(c_j))\ \mathbf{F}\ \neg U_a(c_j)$ given $i \neq j \in \boldsymbol{N}$		for $\forall a \in \boldsymbol{M}$

The third row finds Type 3-4 UCCAs using Abstraction 2a and produces an output consistent with Table 4-10. The notation employs Linear Temporal Logic (LTL) to describe the different sequences of starting and ending different control actions relative to one another.

Each equation is a sequence of three time periods. Item (a) in the list describes those three periods as they pertain to starting u_a before starting u_b. First, the initial condition treats both u_a and u_b as not provided because they have not yet started. This condition holds ***U**ntil* the second step, represented by LTL temporal operator **U** [207], when u_a is started and, therefore, is now provided. Finally, in some ***F**uture* third step, denoted by LTL operator **F**, u_b is started. As

discussed in Section 4.2.1, no assumption is made that u_a is still provided by the time this last step occurs.

The fourth row in Table 4-14 uses Abstraction 2b to output Type 3-4 UCCAs like those shown in Table 4-12. The cases in that row follow the same three temporal steps defined above in LTL. As such, item (c) in that row represents any one controller c_i ending u_a before any other controller c_j starts u_a.

Each enumerated control combination is encoded into a tuple, $\boldsymbol{C_x} = (\boldsymbol{C_x}, \boldsymbol{U_x}, \boldsymbol{T_x})$, as defined below. By abuse of notation, the abstracted set of multiple controllers or of multiple control actions can be encoded as any single element in $\boldsymbol{C_x}$ and $\boldsymbol{U_x}$. A superscript on $\boldsymbol{C_x}$ (i.e., $\boldsymbol{C_x^{Abs2a,T12}}$), informs which abstraction and type of UCCA is enumerated. For Type 3-4 UCCAs, the order of the elements in vectors $\boldsymbol{U_x}$ and $\boldsymbol{T_x^{34}}$ reflects the temporal sequence.

$\boldsymbol{C_x}$ ≔ vector of controllers involved in enumeration

$\boldsymbol{U_x}$ ≔ vector of control actions paired with $\boldsymbol{C_x}$

$\boldsymbol{T_x^{12}} = \{not\ provide, provide\}$; represents if elements $\boldsymbol{C_x}$ provide elements in $\boldsymbol{U_x}$

$\boldsymbol{T_x^{34}} = \{start, end, \emptyset\}$; applied by $\boldsymbol{C_x}$ to $\boldsymbol{U_x}$

Example: for the multi-UxS system, if $u_a = jam$, the third item of each row in Table 4-14 encodes $\boldsymbol{C_x}$ as Equations (8)-(11).

$$\boldsymbol{C_x^{Abs2a,T12}} = (\{team, team\} \quad \{[jam], [strike \vee track]\} \quad \{0,1\}) \tag{8}$$

$$\boldsymbol{C_x^{Abs2b,T12}} = (\{c_i, \exists c_j\} \quad \{[jam], [jam]\} \quad \{1,1\}) \tag{9}$$

$$\boldsymbol{C_x^{Abs2a,T34}} = (\{team, team\} \quad \{[jam], [strike \vee track]\} \quad \{end, start\}) \tag{10}$$

$$\boldsymbol{C_x^{Abs2b,T34}} = (\{c_i, \exists c_j\} \quad \{[jam], [jam]\} \quad \{end, start\}) \tag{11}$$

$\boldsymbol{C^{Abs}}$ denotes the set of all $\boldsymbol{C_x}$ created in the above enumeration. A MATLAB prototype developed to demonstrate this concept automatically produces four tables with the control combinations and placeholders for an analyst to enter potential contexts and other information described in the next step of the Algorithm. Examples of UCCA tables created using the automation are shown in Appendix 2.

Lines 2-4: Human-Identified Context of UCCAs

Once the automation has produced all the potential UCCAs, a human analyst determines for each one (Line 2) if there is a context, or multiple contexts, in which that control combination is hazardous (Line 3). As part of this, the analyst also traces each context to the hazard(s) the UCCA leads to. The context may describe the violation of safety constraints previously identified in the analysis. This part of the algorithm requires human expertise and intuition across multiple domains.

While some contexts can be formally described so that they are automatically found, it is challenging to scale that up beyond simple problems. In addition, the context may involve the environment the system interacts with, which is too unpredictable to fully describe formally [170].

The same concerns have limited the adoption of formal methods in certification (see Chapter 2.3). Humans, as creative and critical thinkers, are better suited for this step.

If a hazardous context is found, it is appended to the control combination (Line 4). For UCCAs defined using Abstractions 2a, the analyst also specifies vector $\boldsymbol{U_x^{rel}}$ to designate the control actions in u_a and u_b that are relevant to the context. Abstractions 2b UCCAs only include one control action and therefore do not require $\boldsymbol{U_x^{rel}}$ to be specified.

Lines 5-6: Automated Refinement of UCCAs

After the human specifies the context(s), the hazard traceability, and the relevant control actions, the automation reads in each unique abstracted UCCA as tuple $\boldsymbol{\mathcal{U}_x^{abs}} = \left(\boldsymbol{C_x}, \boldsymbol{U_x}, \boldsymbol{T_x}, context_x, \boldsymbol{U_x^{rel}}\right)$ (Line 5). The potential UCCAs that are repeated or not considered unsafe are removed from further consideration.

Example: for the multi-UxS system, the $\boldsymbol{\mathcal{U}_x^{abs}}$ encoded for abstracted UCCA 4 specified in Section 4.2.3 is shown in Equation (12). In this case, only the *track* and *strike* commands are relevant to the context.

$$\boldsymbol{\mathcal{U}_x^{abs}} = (\{team, team\} \quad \{[track], [jam \vee strike]\} \quad \{0,1\} \quad \{'when \ldots'\} \quad \{track, strike\}) \quad (12)$$

Next, Algorithm 1 calls a function to refine each abstracted UCCA (Line 6). Here, the automation finds every combination of specific controllers that can contribute to the collective control output in the abstracted UCCA.

For Type 1-2 UCCAs, the tool first considers all the possible control combinations $\boldsymbol{\mathcal{P}} = f(\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{A}, \boldsymbol{T})$ by listing every combination of every controller $\boldsymbol{N}$, with every control action $\boldsymbol{M}$ it has the authority $\boldsymbol{A}$ to provide it, and the options $\boldsymbol{T}$ for providing or not providing that action. Each combination is represented by tuple $\boldsymbol{\mathcal{P}_e} = \left(\boldsymbol{C_e}, \boldsymbol{U_e}, \boldsymbol{T_e^{T12}}\right)$ using the same format as $\boldsymbol{C_x^{T12}}$.

Example: for the multi-UxS system, one $\boldsymbol{\mathcal{P}_e}$ is shown in Equation (13). To shorten the equation, the *jam*, *strike*, and *track* commands are listed a *j*, *s*, *t* respectively. By abuse of notation, each controller in $\boldsymbol{C_e}$ is reiterated for each element in the vectors the controller is matched with with in both $\boldsymbol{U_e}$ and $\boldsymbol{T_e^{T12}}$. In other words, $\{UAS_1, UAS_2, robot\}$ should be interpreted here as $\{[UAS_1, UAS_1, UAS_1], [UAS_2, UAS_2, UAS_2], [robot, robot, robot]\}$.

$$\boldsymbol{\mathcal{P}_e} = (\{UAS_1, UAS_2, robot\} \quad \{[j,s,t], [j,s,t], [j,s,t]\} \quad \{[0,0,1], [0,1,0], [0,1,0]\}) \quad (13)$$

The function then looks for equivalence of the union of control actions provided by all controllers in $\boldsymbol{\mathcal{P}_e}$ to the relevant control actions specified in the abstracted UCCA $\boldsymbol{\mathcal{U}_x^{abs}}$**.** This step is simplified using the earlier assumption that the context of the UCCA is agnostic to which controllers issue the control actions. Equivalence can therefore be determined using subsets $\boldsymbol{\mathcal{P}_e'} = (\boldsymbol{U_e}, \boldsymbol{T_e})$ and $\boldsymbol{\mathcal{U}_x^{abs\prime}} = (\boldsymbol{U_x}, \boldsymbol{T_x})$, which remove the controllers, the contexts, and the irrelevant control actions from consideration. In simple terms, if a relevant control action is provided in the abstracted Type 1-2 UCCA, equivalence is achieved if any controller in the refined set provides that control action.

Example: for the multi-UxS system, Table 4-9 refines a UCCA in which the team provides ¬*track* and *strike*. In this case, any combination where no controller *tracks* and one or more controllers *strike* meets this criterion. As such, items 4.4-4.7, which include multiple controllers

providing the *strike* command, are considered equivalent to the collective team output. The associated $\boldsymbol{\mathcal{U}_x^{abs\prime}}$ and equivalent $\boldsymbol{\mathcal{P}_e'}$ defined for item 4.4 are listed in Equations (14)-(15).

$$\boldsymbol{\mathcal{U}_x^{abs\prime}} = (\{jam, strike\} \quad \{0,1\}) \tag{14}$$

$$\boldsymbol{\mathcal{P}_e'} = (\{[j,s],[j,s],[j,s]\} \quad \{[0,0],[0,1],[0,1]\}) \tag{15}$$

However, *special interactions* may also be defined to influence how equivalence is determined given the context of the UCCA. These interactions may reduce or expand the set of combinations that are considered equivalent. For instance, if multiple controllers provide the *jam* command, they may interfere with each other and yield a collective output equivalent to no *jam* being provided. In this case, if a high-level UCCA states the team provides ¬*jam* and *strike*, any refined UCCA that involves both UAS_1 and UAS_2 providing *jam* and any controller providing *strike* is treated as equivalent. This example would expand the set of combinations to consider.

The special interaction is defined as a tuple $\boldsymbol{\mathcal{S}} = (\boldsymbol{U}, \boldsymbol{T_{in}}, \boldsymbol{\Sigma}, \boldsymbol{T_{out}})$, where $\boldsymbol{\Sigma}$ specifies the rules that are applied to the control actions in $\boldsymbol{U}$ and whether or not they are provided $\boldsymbol{T_{in}}$. The rules indicate how to treat the collective output of those commands using items in vector $\boldsymbol{T_{out}}$. In the *jam* example above, $\boldsymbol{\Sigma}$ includes the rule that if the number of controllers providing the *jam* control action in $\boldsymbol{U}$ exceeds 1, treat *jam* collectively as $\boldsymbol{T_{out}^{12}} = not\ provide$.

Example: for the multi-UxS system, according to this special interaction, the $\boldsymbol{\mathcal{P}_e'}$ defined in Equation (16) would be considered equivalent to the $\boldsymbol{\mathcal{U}_x^{abs\prime}}$ previously listed in Equation (14).

$$\boldsymbol{\mathcal{P}_e'} = (\{[j,s],[j,s],[j,s]\} \quad \{[1,0],[1,1],[0,1]\}) \tag{16}$$

The output of this part of the algorithm is a set of refined control combinations $\boldsymbol{\mathcal{C}_x^{ref}}$ that are considered equivalent to the abstracted UCCA $\boldsymbol{\mathcal{U}_x^{abs}}$. $\boldsymbol{\mathcal{C}_x^{ref}}$ is evaluated using Equation (17) for Type 1-2 UCCAs, which is the mathematical representation of the concepts described above.

$$\boldsymbol{\mathcal{C}_x^{ref,T12}} = \bigcup_{\boldsymbol{e}} \boldsymbol{\mathcal{P}_e} \mid \left(\boldsymbol{\mathcal{U}_x^{abs\prime}} \equiv \bigcup_{i} \boldsymbol{\mathcal{P}_e'} \mid \boldsymbol{\mathcal{S}}, \forall i \in \boldsymbol{N} \right), \forall e \in \boldsymbol{\mathcal{P}} \tag{17}$$

Example: for the multi-UxS system, the Equation (18) shows $\boldsymbol{\mathcal{C}_x^{ref,T12}}$ represented by Table 4-9 and repeated in the discussion below in Table 4-15. The same abuse of notation used in Equation (13) applies.

$$\boldsymbol{\mathcal{C}_x^{ref,T12}} = \begin{vmatrix} (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,0],[0,0],[0,1]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,0],[0,1],[0,0]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,0],[0,0]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,0],[0,1],[0,1]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,0],[0,1]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,1],[0,0]\}) \\ (\{UAS_1, UAS_2, robot\} & \{[j,s],[j,s],[j,s]\} & \{[0,1],[0,1],[0,1]\}) \end{vmatrix} \tag{18}$$

Type 3-4 UCCAs are refined using a similar process. For Abstraction 2a, every possible combination of any one controller, with the proper authority, issuing each of the relevant control actions in the UCCA is enumerated. For Abstraction 2b, every possible combination of one controller issuing the earlier and the later common command signal is found. In both cases, each

enumerated item is labeled with *start* or *end* as designated by $\boldsymbol{T_x^{34}}$ in $\boldsymbol{\mathcal{U}_x^{abs}}$. The terms are rearranged into set $\boldsymbol{\mathcal{C}_x^{ref,T34}}$ using similar conventions as in $\boldsymbol{\mathcal{C}_x^{ref,T12}}$.

Example: for the multi-UxS system, Equations (19) and (20) represent, respectively, $\boldsymbol{\mathcal{U}_x^{abs}}$ and $\boldsymbol{\mathcal{C}_x^{ref,T12}}$ for the unsafe sequence that involves ending *jam* (j) before starting *strike* (s) and *track* (t). To shorten the representation, UAS_1, UAS_2, *robot*, *start* and *end* are listed as U_1, U_2, r, S and E respectively. Furthermore, the backets around the *jam* $[j]$ and its *end* $[E]$ or null $[\emptyset]$ are not shown, but are implied in Equation (20). The same abuse of notation described in Equation (13) applies in Equation (20).

$$\boldsymbol{\mathcal{U}_x^{abs}} = (\{team, team\} \quad \{[j], [s \vee t]\} \quad \{end\ (E), start\ (S)\} \quad \{'when \ldots'\} \quad \{j, s, t\}) \tag{19}$$

$$\boldsymbol{\mathcal{C}_x^{ref,T34}} = \begin{vmatrix} (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[S,S]],[\emptyset,[\emptyset,\emptyset]],[\emptyset,[\emptyset,\emptyset]]\}) \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[S,\emptyset]],[\emptyset,[\emptyset,S]],[\emptyset,[\emptyset,\emptyset]]\}) \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[S,\emptyset]],[\emptyset,[\emptyset,\emptyset]],[\emptyset,[\emptyset,S]]\}) \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[E,[\emptyset,S]],[\emptyset,[S,\emptyset]],[\emptyset,[\emptyset,\emptyset]]\}) \\ \vdots & \vdots & \vdots \\ (\{U_1, U_2, r\} & \{[j,[s,t]],[j,[s,t]],[j,[s,t]]\} & \{[\emptyset,[\emptyset,\emptyset]],[\emptyset,[\emptyset,\emptyset]],[E,[S,S]]\}) \end{vmatrix} \tag{20}$$

In this work, Type 3-4 UCCAs do not employ special interactions $\boldsymbol{\mathcal{S}}$ to specify how different refined sequences of actions should be considered equivalent to the collective sequencing. Such interactions were not considered to be relevant to the problems analyzed as part of this research. However, if a need arises to include such special considerations, a version of Equation (17) that applies to Type 3-4 UCCAs could be derived. That is beyond the scope of this work.

Lines 7: Automated Pruning of Additional Equivalent Combinations

In the development of loss scenarios, which occurs later in the hazard analysis, causal factors are analyzed to explain how the control combinations in the UCCA may occur. This process needs the refined UCCAs to consider what controllers issued what control actions, as different controllers may have different causal factors. However, some of the controllers may be considered *interchangeable* from a scenario perspective. In such cases, the additional refined UCCAs that lead to the same causal analysis must be eliminated to avoid duplication of effort.

In the multi-UxS example (Figure 4-13), the engineering team may consider UAS_1 and UAS_2 to be interchangeable. Causal scenarios will be no different if it is UAS_1 that provides part of the UCCA, or instead, UAS_2. As a result, items 4.2 and 4.3 in Table 4-15 are equivalent as they both involve one UAS providing the *strike* command. No additional information is gained from 4.3 if 4.2 is analyzed. As such, UCCA 4.3 is pruned from the set.

Similarly, items 4.4 and 4.5 both involve one UAS and the robot providing the *strike* command and, therefore, 4.5 is pruned because it is duplicative. In contrast, items 4.2 and 4.5 are not equivalent even though they alternate the one UAS that provides *strike*. In item 4.2, UAS_2 provides the *strike* alone, while in 4.5, UAS_1 and the robot provide the *strike*.

Table 4-15. Pruning and Prioritizing Combinations in Refined UCCA 4

UCCA	Team	Team	Context
4	¬track	{~~jam~~ ∨ strike}	*when enemy must be tracked as strike occurs [H3]*

Refined	UAS_1	UAS_2	robot	Pruned?	Priority
4.1	¬track	¬track	strike	No	High
4.2	¬track	strike	¬track	No	High
4.3	strike	¬track	¬track	Yes (4.3 ≡ 4.2)	N/A - Pruned
4.4	¬track	strike	strike	No	Low
4.5	strike	¬track	strike	Yes (4.5 ≡ 4.4)	N/A - Pruned
4.6	strike	strike	¬track	No	Low
4.7	strike	strike	strike	No	Low

One of the inputs to Algorithm 1 is set $\boldsymbol{C^{int}}$ that contains z vectors of *interchangeable controllers*. In Line 7, a function uses this input to prune the equivalent duplicated control combinations from the set $\boldsymbol{C_x^{ref}}$ generated in Line 6. The process takes any two combinations $\boldsymbol{C_{x,m}^{ref}}$ and $\boldsymbol{C_{x,n}^{ref}}$ from $\boldsymbol{C_x^{ref}}$, checks if the vector of control efforts between any two interchangeable controllers are equivalent, and also checks that all control efforts by the other controllers are consistent. These conditions are captured in Equation (21), and if met, then the UCCAs are equivalent and $\boldsymbol{C_{x,n}^{ref}}$ is pruned from $\boldsymbol{C_x^{ref}}$.

$$\boldsymbol{C_{x,m}^{ref}} \equiv \boldsymbol{C_{x,n}^{ref}} \mid \left([U_m(c_i)] = [U_n(c_j)]\right) \wedge \left([U_m(c_j)] = [U_n(c_i)]\right) \wedge \left([U_m(c_k)] = [U_n(c_k)]\right), \quad \forall m \in \boldsymbol{C_m}, \forall n \in \boldsymbol{C_n}, i \neq j \in \boldsymbol{C_z^{int}}, \forall k \notin \boldsymbol{C_z^{int}}, \forall z \in \boldsymbol{C^{int}} \tag{21}$$

Example: for the multi-UxS system, UAS_1 and UAS_2 are interchangeable as defined by $\boldsymbol{C^{int}}$ in Equation (7), and one possible $\boldsymbol{C_x^{ref}}$ is represented by Equation (18). The two items $\boldsymbol{C_{x,m}^{ref}}$ and $\boldsymbol{C_{x,n}^{ref}}$, respectively shown in Equations (22) and (23), are members of $\boldsymbol{C_x^{ref}}$. They represent items 4.2 and 4.3 described in the same example above and shown in Table 4-15.

$$\boldsymbol{C_{x,m}^{ref,T12}} = (\{UAS_1, UAS_2, robot\} \quad \{[j,s],[j,s],[j,s]\} \quad \{[0,0],[0,1],[0,0]\}) \tag{22}$$

$$\boldsymbol{C_{x,n}^{ref,T12}} = (\{UAS_1, UAS_2, robot\} \quad \{[j,s],[j,s],[j,s]\} \quad \{[0,1],[0,0],[0,0]\}) \tag{23}$$

Here, $c_i = UAS_1 \in \boldsymbol{C^{int}}$, $c_j = UAS_2 \in \boldsymbol{C^{int}}$, $c_k = robot \notin \boldsymbol{C^{int}}$. The components of Equation (21) for this example are captured in Equations (24)-(27), and the three conditions necessary to prune $\boldsymbol{C_{x,n}^{ref,T12}}$ are met in Equation (28).

$$[U_m(c_i)],\ \forall m \in \boldsymbol{C_m} = [U_m(UAS_1)], u_m \in [jam, strike] = [0,0]; \tag{24}$$

$$[U_m(c_j)],\ \forall m \in \boldsymbol{C_m} = [U_m(UAS_2)], u_m \in [jam, strike] = [0,1]; \tag{25}$$

$$[U_m(c_k)],\ \forall m \in \boldsymbol{C_m} = [U_m(robot)], u_m \in [jam, strike] = [0,0]; \tag{26}$$

$$[U_n(c_i)],\ \forall n \in \boldsymbol{C_n} = [1,0];\ \ [U_n(c_j)],\ \forall n \in \boldsymbol{C_n} = [0,0];\ [U_n(c_k)],\ \forall n \in \boldsymbol{C_n} = [0,0]; \tag{27}$$

$$[U_m(c_i)] = [U_n(c_j)]; \qquad [U_m(c_j)] = [U_n(c_i)]; \qquad [U_m(c_k)] = [U_n(c_k)] \tag{28}$$

Conditions to specify controllers as *interchangeable* may vary by application. The taxonomy introduced in Figure 3-2 can help reason about what controllers lead to similar causal scenarios. It describes the structure of the interactions between controllers, which influences causality and the system dynamics. Commonality in some of the axes may help determine interchangeability.

As an example, consider the arbitrary collaborative system in Figure 4-15, in which five controllers share a process. The controllers are differentiable on at least the first two dimensions of the taxonomy. The interactions involved are human-human, human-machine, and machine-machine. The hierarchal structure includes supervisory control and peer interactions. Based on the controller types and hierarchy, the set of interchangeable controllers may be specified as $\boldsymbol{C^{int}} = \{\{human_1, human_2\}, \{machine_1, machine_2\}\}$. The other five dimensions in the taxonomy could also be included in this consideration if of interest to the analysis.

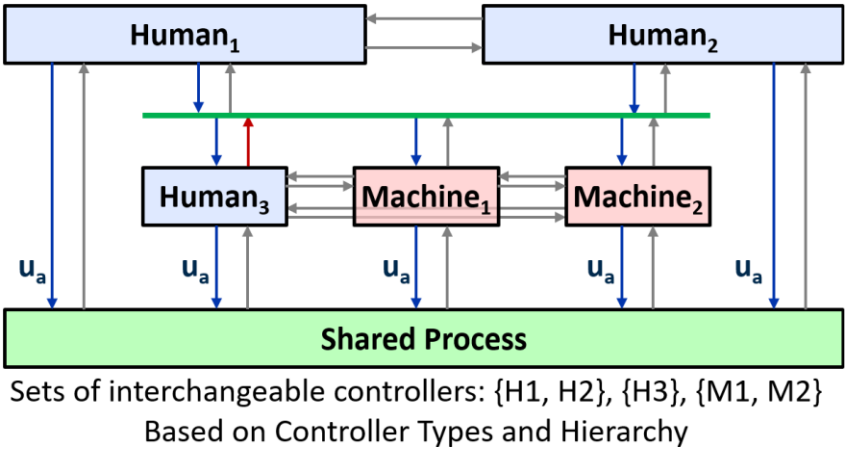


Figure 4-15. Interchangeable Controllers in Arbitrary Collaborative System

Lines 8-9: Automated Prioritization and Output of UCCAs

The refined UCCAs may optionally be prioritized to focus the remainder of the hazard analysis (Line 8). The goal is to highlight the UCCAs that will contribute more new information in causal scenario analysis and devalue those that provide more repetitive information. The prioritization scheme implemented in this dissertation is a heuristic formed on engineering judgment, but it could be implemented differently based on other application needs.

The concept is illustrated using again the ¬*track* and *strike* UCCA from the multi-UxS example refined in Table 4-15. Scenarios for this UCCA must explain why (1) any controller would provide *strike* when (2) no controller provides *track*, and vice versa. It is arguably less important in the context of the UCCA to focus on why multiple controllers would provide *strike*. As such, in the remaining set of refined UCCAs that were not pruned, those with one controller providing *strike* are prioritized over those with multiple controllers issuing that command.

It is also important to note, in this example, that if there is a context in which multiple controllers providing *strike* is unsafe, that interaction is specifically addressed in Abstraction 2b, where combinations of controllers provide the same action (e.g., $c_i\ strike\ \wedge\ \exists c_j\ strike$). For this reason, the prioritization scheme does not devalue combinations of two controllers providing the action in Abstraction 2b. However, using a similar logic as above, it does deprioritize combinations of three or more controllers providing the same action, as those may be repetitive.

In this dissertation, the prioritization scheme is also employed to devalue UCCAs in which all the control actions are provided by one controller. For example, if commands u_1 and u_2 are unsafe together, and if controller c_1 is the only one to provide those two commands, the UCCA is deprioritized. Such instances are arguably less related to collaborative control. However, other applications may choose to treat such occurrences with higher priority.

The special interactions specified in $\boldsymbol{S}$ can also be used to influence prioritization. In the example previously used to illustrate $\boldsymbol{S}$, multiple controllers providing the *jam* command are treated collectively as not providing *jam*. Here, $\boldsymbol{S}$ ensures that cases of two controllers providing *jam* in a UCCA are not deprioritized. However, by the same reasoning provided for Abstraction 2b UCCAs above, those with any three controllers providing *jam* are deprioritized.

Because the assumptions underpinning prioritization are softer than those used in the pruning process, the deprioritized UCCAs are not eliminated. UCCAs are presented to the analyst by order of priority, and the option remains to analyze those labeled as lower in priority in scenario development.

The set of UCCAs refined, pruned, and prioritized using automation is returned by Algorithm 1 as set $\boldsymbol{U^{ref}}$ (Line 9). An example of the output produced by the prototyped automation tool is provided in Appendix 2. The UCCAs, both abstracted and refined, are now ready for the analyst to proceed to the last step in the hazard analysis: the development of causal scenarios.

4.3 Causal Scenarios in Collaborative Control

The fourth and final step in STPA develops loss scenarios to identify causal factors that can lead to the unsafe control actions (UCAs) [50]. Safety constraints can then be specified to guide the design and operation of a system to eliminate or control these factors to prevent losses.

In STPA, scenarios are identified by analyzing each UCA to determine (1) why the controller would provide the UCA and (2) why a control action would be improperly executed or not executed leading to the outcome of the UCA. The process explores potential breakdowns in four different parts of a feedback control loop. The STPA guidance describes common factors to consider in each part to assist in the analysis (see Figure 4-16).

An approach proposed by Thomas aims to add structure and enhance the ability to build scenarios top-down [208]. The process starts by formulating four *basic scenarios*, generically listed below, which originate from the four parts of the feedback control loop. The basic scenarios are then further refined as necessary to develop safety constraints to mitigate the associated factors.

1. Basic Scenario 1: Unsafe Controller Behavior: the controller receives adequate feedback, but still makes unsafe decisions.
2. Basic Scenario 2: Unsafe Feedback or Other Information: the controller receives inadequate feedback leading to an unsafe decision.
3. Basic Scenario 3: Unsafe Control Path: the controller provides a safe control action, but the controlled process receives a control action that is unsafe.
4. Basic Scenario 4: Unsafe Process Behavior: the safe control action is received by the controlled process, but the process behaves in an unsafe way. [208]

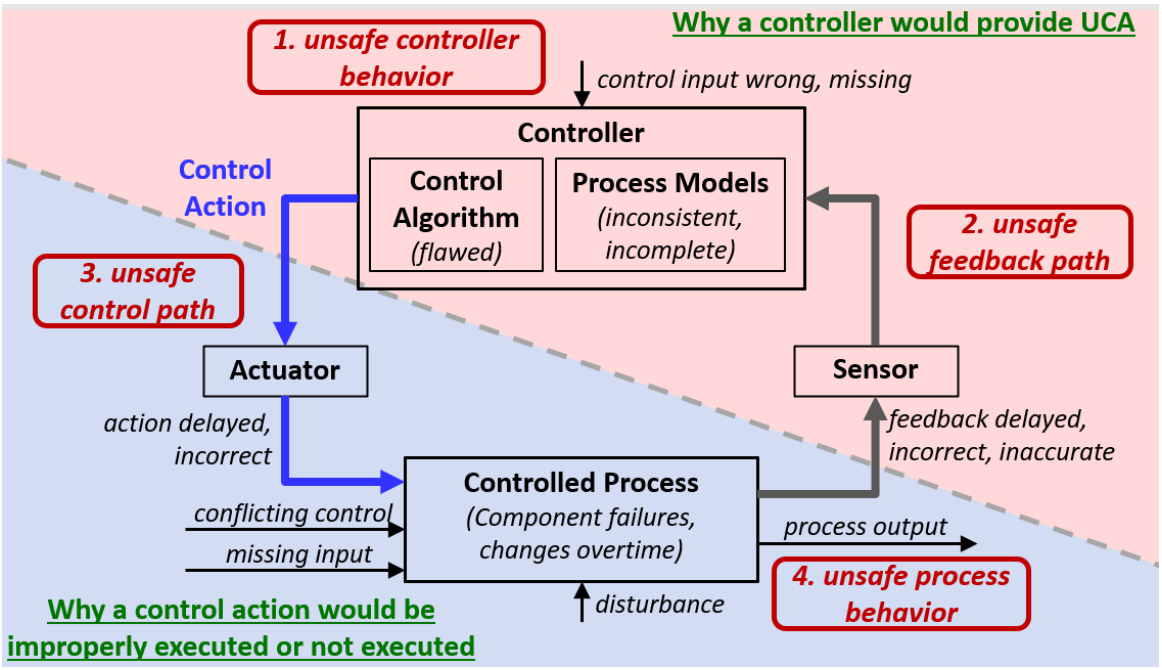


Figure 4-16. Areas of Potential Breakdown in a Feedback Control Loop (derived from [50])

The scenario development process in STPA is able to uncover causal factors not found by other hazard analysis techniques [209]. However, the approach focuses on one unsafe control action provided by one controller at a time. As described in Section 4.2, some causal factors may only be identified by exploring how multiple control actions are unsafe together.

Now that unsafe combinations of control actions (UCCAs) are identified, a process outlined in Figure 4-17 is introduced to develop causal scenarios from these UCCAs. The process has the following goals. First, it aligns with STPA by considering both (1) why unsafe control actions would be provided and (2) why control actions would not be properly executed. Second, it provides a mechanism to analyze the multiple feedback control loops involved in the UCCAs collectively. Third, it accounts for the collaborative control dynamics defined in Chapter 3. And fourth, the approach is systematic in scenario identification and refinement.

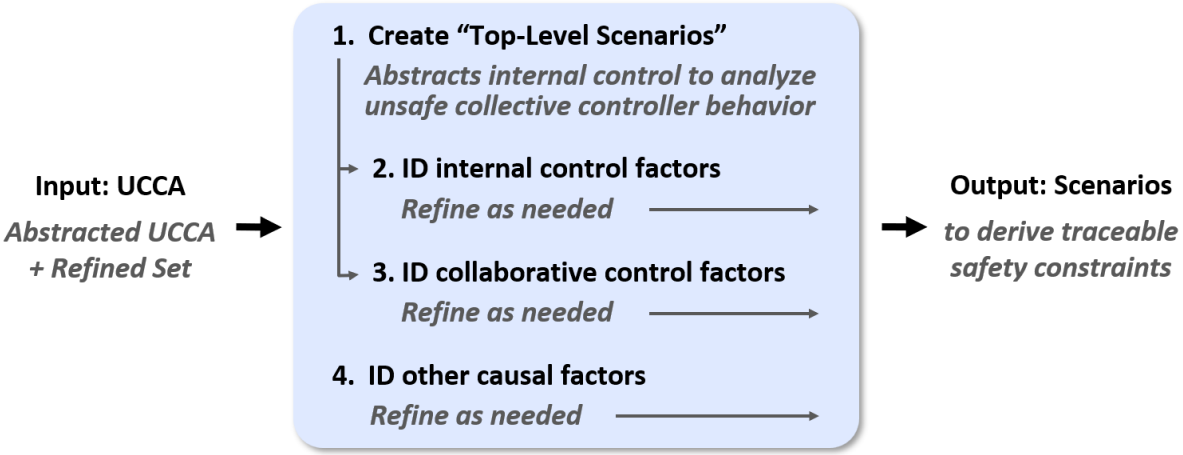


Figure 4-17. Process to Develop Causal Scenarios from a UCCA

The input to the process is a UCCA identified using the technique in Section 4.2. Each UCCA represents an unsafe collective control output by the team to the shared process. As in STPA, the scenario identification process explores factors that lead to unsafe behaviors of the team of controllers, its feedback paths, its control paths, and the controlled process. However, this work emphasizes how the interactions among the controllers on the team contribute to unsafe collective team behavior. The analysis of the feedback paths, control paths, and controlled processes follows the same reasoning as in STPA. This key idea is illustrated in Figure 4-18.

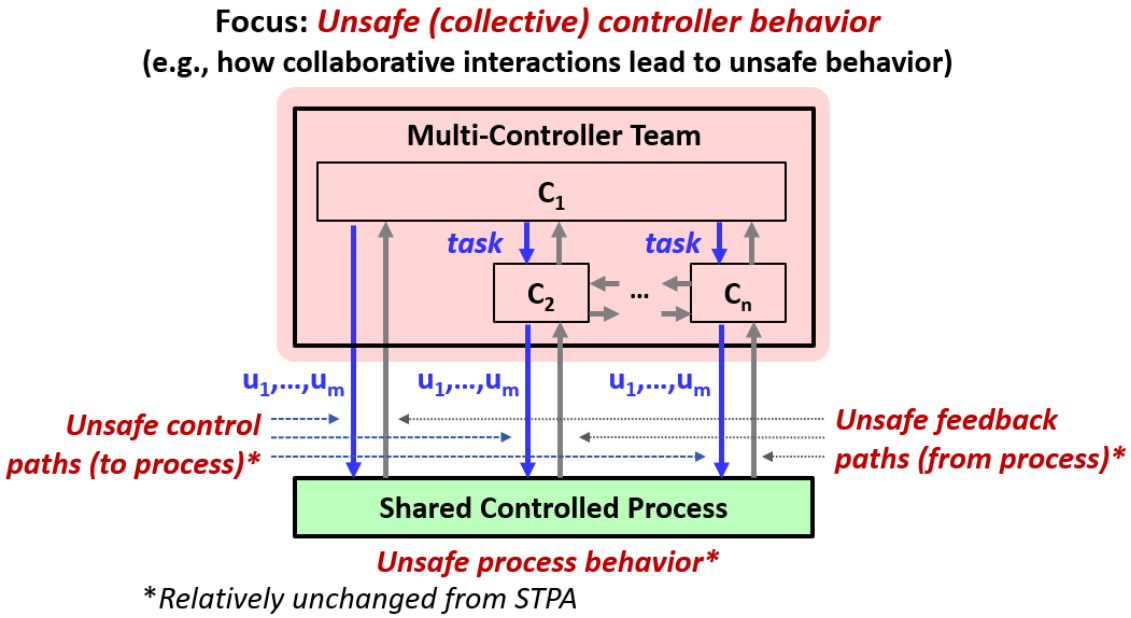


Figure 4-18. Four Areas of Potential Breakdown in Multiple Feedback Control Loops

In Step 1, the scenario identification process (Figure 4-17) explores how hierarchal control within the team contributes to the unsafe collective controller behavior in the UCCA. To illustrate this concept, consider the system in Figure 4-18, where C_1 has control authority over the other controllers on the team in addition to the shared process. The first step considers how the different potential control actions from C_1 to the other controllers relate to the UCCA.

In simple terms, this step investigates how the UCCA could occur if, for example, C_1 commands the team to provide an unsafe output, or as another example, if the other controllers do not properly execute commands from C_1. Each example represents a new scenario to consider. A method defined in Section 4.3.1 helps to systematically account for the different possible internal control actions that lead to unsafe collective team behavior using *top-level scenarios*.

Each of these scenarios is then iteratively refined using a template introduced in Section 4.3.2. In Step 2, the template finds causal factors in the control loops internal to the team. In Step 3, the template identifies factors related to the collaborative control dynamics of the team.

Finally, Step 4 identifies factors that relate to unsafe feedback paths from the controlled process, unsafe control paths to the controlled process, and unsafe controlled process behavior. These items follow the same approach as used in STPA.

The output of the process is a set of causal scenarios from which engineers can derive safety constraints to eliminate or mitigate the factors that lead to hazards. The remainder of the section

describes each step of the process in more detail. Examples of its application and of the safety constraints that are derived from it are provided in the Chapter 5 case study.

4.3.1 Step 1: Top-Level Scenarios to Reason about Internal Control

Step 1 of the scenario identification process examines how the collective output in a UCCA relates to the different possible control actions internal to the team. To illustrate this concept, assume that the system shown in Figure 4-18 has a UCCA that involves multiple controllers providing control action u_1. This case occurs in the multi-UxS example in Section 4.2, when multiple controllers provide the *jam* command (see Table 4-11), and it is a Type 1-2 UCCA.

Using the process described in Section 4.2.4, the two priority refined UCCAs, in this case, are (1) c_1 and c_2 both provide u_1, and (2) c_2 and c_n both provide u_1. Step 1 now explores how the commands provided by c_1 to $\{c_2, \ldots, c_n\}$ may have contributed to these outcomes. Assuming c_1 can task the other controllers to execute u_1, Figure 4-19 enumerates the four different possible c_1 internal control actions that are relevant to each of the two refined UCCAs.

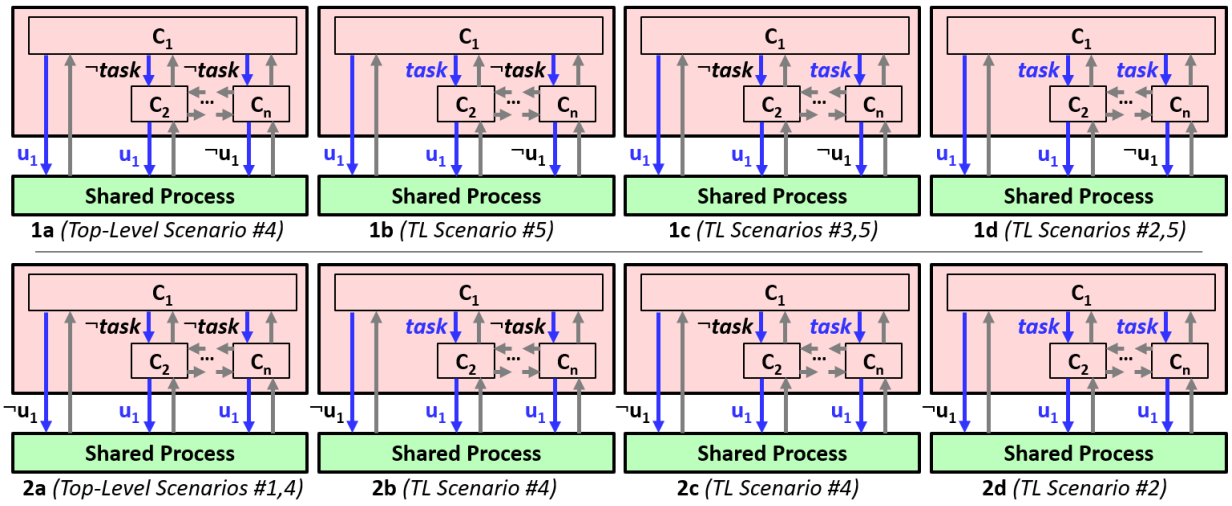


Figure 4-19. Possible Internal Control Actions that Can Lead to Type 1-2 UCCA

Each different set of internal control actions represents a new potential scenario to analyze. For instance, in item 1a, c_1 does not command any other controller to provide u_1, and instead, provides the command itself. Yet, c_2 also issues the command despite not being tasked. This forms a scenario that can be further refined to explain the unsafe behavior of the team as a whole.

Reasons for this outcome may include c_1 unintentionally providing the task, c_2 receiving the command from another controller, c_2 unintentionally providing the command, and so on. A systematic process to explore these causal factors is introduced in the next section. The other items in Figure 4-19 may also be developed into additional scenarios.

While this simple example only involves one type of control action (u_1), it still results in eight potential scenarios to further analyze. This number grows exponentially with the number of different types of internal control actions involved $\{u_1, \ldots, u_n\}$. Therefore, a full enumeration with any more complexity can produce an intractable number of scenarios. This is similar to the problem encountered in UCCA identification. Simplification is again necessary.

Abstraction is again applied to manage the combinatorial complexity. The different possible internal control actions are abstracted into *top-level* scenarios that cover their general concern toward the unsafe collective output. Table 4-16 lists the generic *top-level* scenarios that help reason about how internal control relates to Type 1-2 UCCAs. These scenarios are directly traceable to UCA Types 1 and 2 in STPA and have a similar intent to the *basic scenarios* by Thomas [208].

Table 4-16. Top-Level Scenarios that Address Internal Control Issues for Type 1-2 UCCAs

#	Top-Level Scenario	Full Top-Level Description
1	Direction Not Provided (Unsafe)	A controller does not direct other controllers on the team as necessary for the team to execute safe collective control of the shared process.
2	Direction Provided (Unsafe)	A controller directs other controllers on the team in a way that leads to unsafe collective control. Includes: directing wrong controller to provide command, directing multiple controllers in a way that conflicts with one another, and directing controller to provide incorrect command
3	Direction Provided (Safe) but Not Executed Properly (Unsafe)	A controller directs other controllers on the team adequately, but some of those controllers do not execute directions properly, which leads to unsafe collective control. Includes: controllers do not provide some commands, controllers provide commands improperly, wrong controller provides command
4	Direction Not Provided (Safe) but Executed (Unsafe)	A controller adequately does not direct other controllers on the team to provide certain commands, but some of those controllers provide them anyways, which leads to unsafe collective control.
5	Controller Actions to Process and Directions it Provides (Unsafe)	A controller provides control actions to the shared process that are unsafe in combination with how it directs other controllers on the team. Includes: improperly providing a control action that is necessary in combination with directed actions, providing a control action that conflicts with directed actions

While the top-level scenarios are designed to provide coverage over the different possible internal control combinations, they are not mutually exclusive of one another. Figure 4-19 illustrates this point by mapping each control combination into these scenarios. Some of the cases fit into multiple scenarios. The analytical overlap is intentional to avoid potential gaps.

For example, in item 1c, c_1 tasks a specific controller to provide u_1, but a different controller provides it instead. This situation fits into top-level scenario #3 in Table 4-16. In addition, despite tasking another controller to provide u_1, c_1 also provides the command itself, which is captured by top-level scenario #5. The scenarios help systematically consider these issues and are then further refined in the context of the UCCA, as described in the next subsection.

As defined in Chapter 3, *dynamic hierarchy* occurs when multiple controllers can mutually command each other. As such, *dynamic hierarchy* contributes to the possible control actions

internal to the team. For this reason, this interaction is addressed in the causal analysis using the top-level scenarios defined above. Multiple instances of each of the scenarios can be created to cover changes in the controller, providing directions to others on the team.

For example, the Pilot - Digital Copilot system surveyed in Chapter 3.3 exhibits dynamic hierarchy in collaborative checklist execution. Top-Level Scenario #1 can explore how the pilot does not direct the automation to execute certain functions and vice-versa, it can also examine how the automation does not direct the pilot. Examples of scenario development related to dynamic hierarchy are provided in the case study in Chapter 5.

Type 3-4 UCCAs, which describe how starting and ending control actions relative to one another are unsafe, face similar combinatorial challenges. For example, consider an unsafe gap in the handoff of control action u_1 between any two controllers in the same system analyzed above. In other words, c_i ends u_1 before c_j starts u_1.

This UCCA refines into the three priority UCCAs shown in Figure 4-20. Each may be related to different command options provided by c_1. In the figure ***F*** is the temporal operator for *some **F**uture step* and designates the later of the control action(s) started (S) or ended (E) in the sequence.

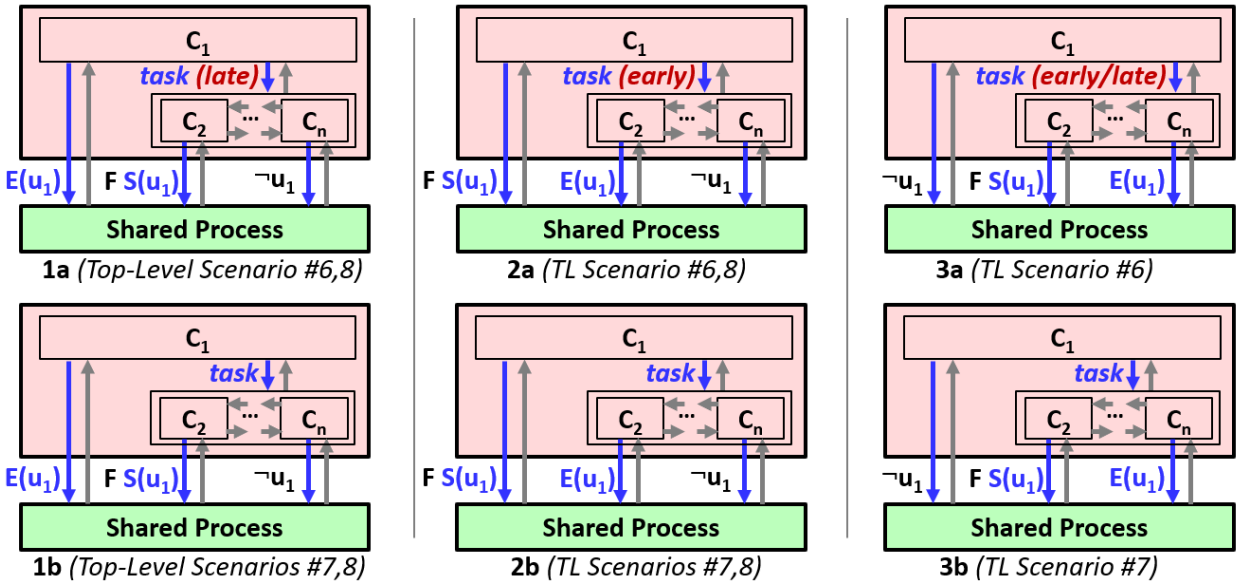


Figure 4-20. Possible Internal Control Actions that Can Lead to Type 3-4 UCCA

Using the same reasoning applied to Type 1-2 UCCAs, Table 4-17 provides the top-level scenarios to abstract the different possible internal controls that lead to Type 3-4 UCCAs. A demonstration of how to tailor top-level scenarios to analyze a system is provided in Chapter 5 for the case study. The next subsection explains how to refine these scenarios in the context of the UCCA.

Table 4-17. Top-Level Scenarios that Address Internal Control Issues for Type 3-4 UCCAs

#	Top-Level Scenario	Full Top-Level Description
6	Directed Sequence Unsafe	A controller directs other controllers on the team in a way that leads to unsafe temporal sequencing.
7	Directed (Safe) but Executed in Unsafe Sequencing	A controller adequately directs other controllers on the team, but the way in which those controllers execute the directions leads to unsafe temporal sequencing.
8	Controller Actions to Process and Directions Unsafe in Sequencing	A controller on the team provides control actions to the shared process that are unsafe in temporal sequencing with how it directs other controllers on the team.

In some systems, the set of collaborating controllers that share authority over a process are all peers and do not have authority to provide control actions to one another. In such cases, top-level scenarios 1-4 and 6-7 are still applicable to explore the different possible combinations of control actions provided to these controllers by supervising controller(s). Top-level scenarios 5 and 8 do not apply if the supervisor(s) do not issue control commands directly to the shared process. The multi-UxS system in Figure 4-13 illustrates this structure, where the UAS and the robot collaborate as peers, and the operator is the supervisor.

4.3.2 Step 2: Internal Control Causal Factors

Steps 2 and 3 in the extended scenario identification process identify causal factors that lead to the unsafe collective controller behavior in each top-level scenario. The template introduced in Figure 4-21, which is a refinement of the process overview in Figure 4-17, provides a systematic approach to consider these causal factors at a high level and then iteratively refine them as necessary. This section and the next introduce the key concepts in this template. Its application is demonstrated in the Chapter 5 case study.

Step 2 of the scenario identification is illustrated by the top yellow box in Figure 4-21. It involves finding, for each top-level scenario, the causal factors associated with feedback control loops internal to the team. The control loops explored are those where a controller provides control actions to the other controllers on the team. In the example shown in Figure 4-19, these are the feedback control loops from c_1 to c_2 and c_1 to c_n.

This step explores how the controller providing the direction (c_1 in the example) may have *unsafe control inputs*, an *inadequate control algorithm*, *inadequate models* of the controllers it is directing ($c_2, \dots, c_n$), and *unsafe control paths* to those controllers. These are the same factors considered in STPA [50]. The only difference is that multiple internal control loops may be considered at once.

The lower yellow box is a refinement template for one of the high-level causal factors. If more detail is needed to explain why the controller providing direction has an inadequate process model of the directed controller, the template points to different reasons for inadequate feedback. For example, c_2 may not send feedback to c_1, or c_1 may not receive the feedback, or c_1 may interpret the feedback incorrectly, and so on. Once again, this guidance is from STPA [50].

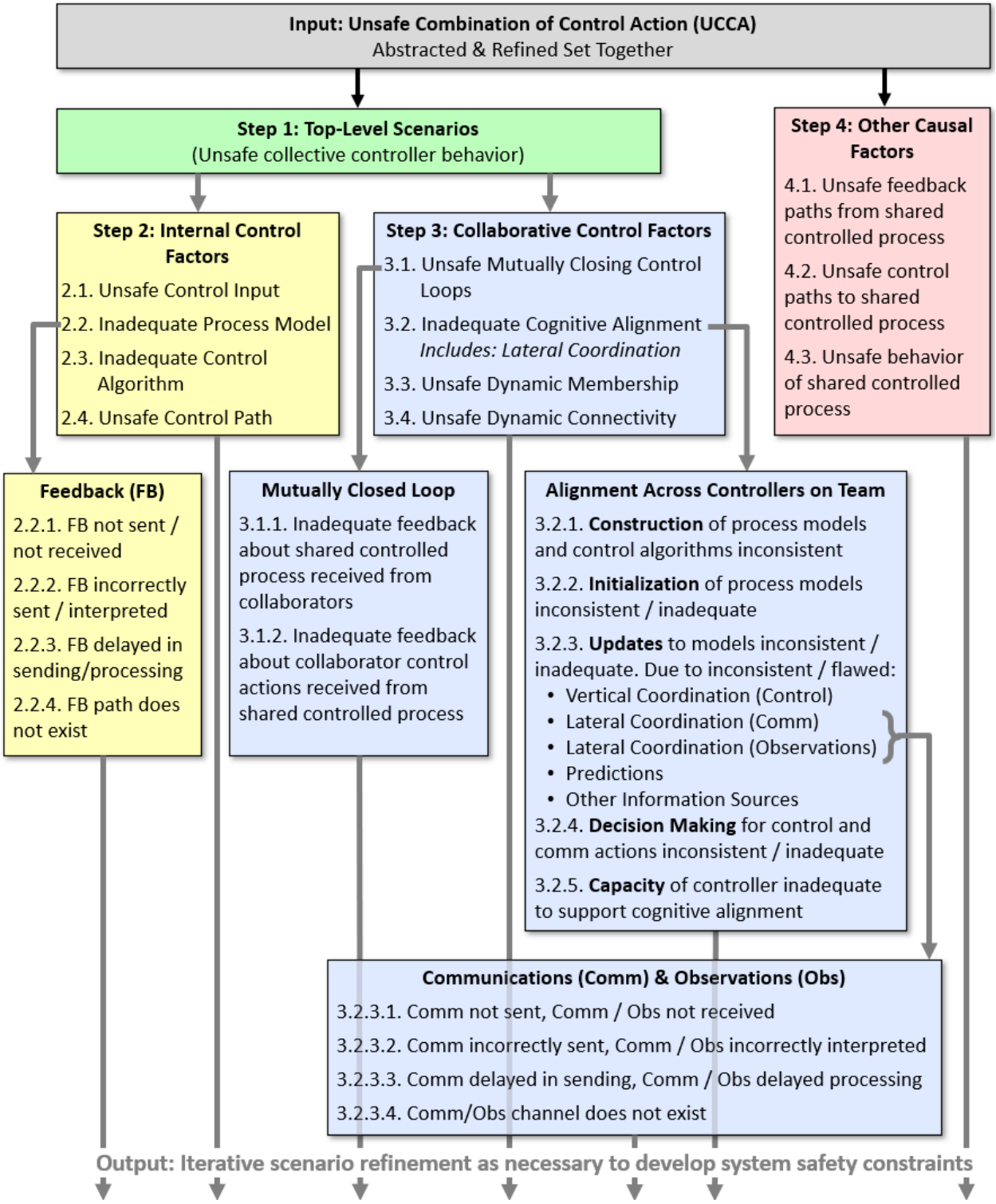


Figure 4-21. Iterative Refinement Template for Causal Scenario Development

4.3.3 Step 3: Collaborative Control Causal Factors

A key goal of this research is to extend the analysis to cover the collaborative control dynamics defined in Chapter 3. Up to this point in the analysis, four of the nine collaborative interactions have already been addressed. As explained in Section 4.2, *shared authority*, *dynamic authority*, and *transfer of authority* are inherently included in the UCCA identification. Similarly, *dynamic hierarchy* is captured in the top-level scenarios that describe internal control (see Section 4.3.1).

The five remaining collaborative control dynamics are addressed in Step 3 of the scenario development process, as illustrated in the top blue box in Figure 4-21. These include *mutually closing control loops*, *cognitive alignment*, *lateral coordination*, *dynamic membership*, and *dynamic connectivity*. The following describes how each is handled.

Mutually Closing Control Loop Causal Factors

In *mutually closing control loops*, feedback control loops are closed across multiple controllers. For scenario development, it is useful to refocus the control structure on this interaction. If in the example system above in Figure 4-18, u_1 and u_2 involve mutually closing loops, Figure 4-22 represents the refocused control structure to explore this dynamic, where "FB" means feedback.

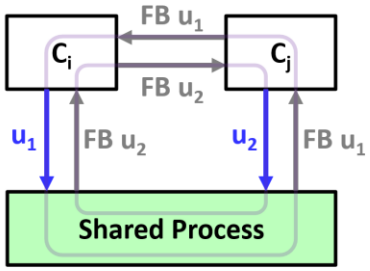


Figure 4-22. Refocused Control Structure for Mutually Closing Control Loops

In the refocused control structure, the controllers are generic (c_i and c_j) to account for different controllers involved in the refined UCCAs. The hierarchy between controllers does not need to be shown, as it is not the focus of this dynamic. The key concept to represent is that c_i senses feedback from a control action provided by c_j, and passes that feedback to c_j, which may otherwise not have access to it. This feedback also influences how c_i controls its part of the process and interactions with c_j.

A scenario refinement template is provided in Figure 4-21 (see the *Mutually Closed Loop* blue box) to further explore causal factors associated with such control loops. The analyst first considers the consequences of a controller receiving inadequate feedback from collaborators regarding its control actions to the shared process. In Figure 4-22, c_j may provide u_2 in an unsafe way if the feedback from c_i is inadequate.

Next, the analyst examines how a controller is influenced by inadequate feedback from the shared process regarding the actions of its collaborators. In the example, c_i may provide u_1 in an unsafe way if inadequate feedback from the process leads it to misinterprets how c_j is controlling the process. Reasons for the inadequate feedback can then be further refined using the same bottom yellow refinement template in Figure 4-21 previously discussed.

Cognitive Alignment and Lateral Coordination Causal Factors

In the *cognitive alignment* dynamic, scenario development focuses on why multiple controllers have process models and make decisions that are inconsistent with one another. A refinement template was developed using ideas synthesized from Thomas [210], France [189], and Johnson [191]. This template is shown in Figure 4-21 (see the *Alignment Across Controllers* blue box). To explore the causal factors in it explores, it is helpful to consider some of the items tracked in the process models of each controller as previously described in Figure 4-6.

The template first accounts for potential differences in how the cognitive functions were constructed. For example, two machines working together may be loaded with different software versions that make them incompatible with one another. Similarly, humans may have been trained differently than their teammates. These same concerns apply to human-machine teams.

The template then studies how process models may be initialized inconsistently across a team. Even if the controllers are aligned in cognitive construction, the different initial conditions available to each can prevent their models from synchronizing. For example, in the Air France 447 accident (see Chapter 1.2), the captain returned from crew rest after the initial aircraft control disturbance [35]. His mental model of aircraft control at that moment was initialized differently than the other two crew members, which contributed to the confusion in the cockpit.

The analysis then examines how the models of the collaborative controllers may be updated inconsistently with one another. Even if the cognitive functions are constructed and initialized similarly, flawed interactions may cause them to drift from one another over time. Several items are considered here. One is inconsistent *vertical coordination* as defined by Johnson [191]. The synchronization of models between multiple teammates may be influenced using hierarchal control. This occurs, for instance, when an Air Traffic Controller (ATC) provides traffic advisories to two merging aircraft.

The next items include communications and observations that lead to flawed *lateral coordination*. These relationships, which are expressed in the collaborative control structure, represent how controllers deliberately and non-deliberately influence each other without using control. For example, pilots of multiple aircraft at an uncontrolled airport laterally coordinate via active communication and passive observations. Inadequate exchange and interpretation of this information can lead models to drift. A template to further refine these factors is shown in the bottom blue box in Figure 4-21.

Here, lateral coordination is included as part of the broader cognitive alignment process because this model fits most of the aerospace systems studied in Chapter 3.3. However, if a system includes lateral coordination but not cognitive alignment, as occurs in some cases, the analyst can still use the lateral coordination refinement template shown in Figure 4-21 separately.

Models may also update inconsistently due to flawed *predictions* made by controllers regarding each other. Controllers often project what the future state of a process will be based on previous knowledge. Many machines employ such predictions in state estimation techniques to overcome noisy, infrequent, or missing feedback [211]. Humans use predictions to estimate where a teammate will be in the future to pass the ball or to determine how a pilot will navigate under lost communications [212]. Flawed predictions can contribute to model drift, especially in the absence of other information.

Lastly, misaligned model updates may result from inconsistencies in other sources of information. These may include differences in observations of common objects, as specified by Johnson [191]. They can also be misaligned feedback received from the shared controlled process or differences in the information received from the environment or controllers beyond the team.

The template for cognitive alignment then examines why *decision-making* may be inadequate as a team. This can occur even if the controllers are synchronized in cognitive construction, model initialization, and model updates. Controllers on a team may have consensus on how they will make a decision, but if it takes them too long to reach that decision or if the decision they collectively reach is incorrect, their output may be unsafe. An example of this is algorithmic churn in distributed systems, when controllers try to reoptimize too frequently based on each other's actions and ultimately do nothing [145].

Finally, the template considers the impact of controller *capacity* on cognitive alignment. As explained in Section 4.1, a controller may be limited by its workload and capabilities. It may not have the capacity to follow commands, coordinate, and make observations, predictions, and decisions that are synchronized with others.

Dynamic Membership and Dynamic Connectivity Causal Factors

The last two collaborative control dynamics shown in Figure 4-21 must be examined. *Dynamic membership* simply explores how a UCCA could occur because controllers come and go on the team. For example, if a controller takes on a task but that controller is subsequently removed from the team, it could affect the actions of collaborators and the collective output. The addition of a new controller or the uncertainty in the team composition may also be causal factors in unsafe control.

Dynamic connectivity considers how expected changes in the team topology can lead to the unsafe collective behavior. This factor is already covered, to some extent, in STPA by examining unsafe control paths and unsafe feedback paths that inhibit information flow. This work extends the consideration to include flawed information relaying, asynchronous information propagation, and temporary disconnects across the team.

Because these two dynamics are conceptually simpler than the others, no dedicated refinement templates were deemed necessary for them in this research. However, such templates can be created and added to the process as necessary in future work.

4.3.4 Step 5: Other Causal Factors

Steps 1, 2, and 3 in the scenario development process aim to explain the unsafe controller behavior of the collective team. As shown in Figure 4-18, the process also considers the other elements in the feedback control loop examined in STPA, including *unsafe feedback paths* from the shared controlled process, *unsafe control paths* to that process, and *unsafe process behavior*.

The analysis here follows the STPA guidance. A controller on the team may have an unsafe model of the shared process because of inadequate feedback it receives from the process. A controller may unintentionally provide or not provide a control action due to a flaw in the control path. Finally, the shared process may adequately receive the collective control inputs from the team but still exhibit unsafe behavior due to other inputs and disturbances.

Some of these factors may already be identified in the scenarios involving collaborative control, which is not a problem. The analytical process favors overlap instead of gaps in consideration.

4.3.5 Final Thoughts on Iterative Refinement Process

The scenario refinement process is demonstrated in the Chapter 5 case study, and a full dataset is available in Appendix 3 and 4. As in STPA, the development of causal scenarios in this work depends on human reasoning and leverages the experience and creativity of the analysis team. The refinement template aims to reduce some of the cognitive burden on human analysts by systematically directing their thought processes across the various factors.

However, the template should not be used as a checklist. The analysts cannot simply hit every item on the list and assume the analysis is complete. Some factors in the template may not be relevant, and others may require careful consideration at multiple levels of abstraction. Furthermore, no claim is made that the template is complete. Other factors beyond those listed may also be uncovered.

Finally, the determination of how much refinement is necessary to complete the analysis remains an open research question. The process encourages iterative refinement so that the high-level scenarios provide as complete coverage as possible. As shown in Chapter 6, the results of the hazard analysis can influence designers to remove complexity from the design to eliminate scenarios at a high level. However, in other cases, the analysis must provide further details to build safety into the design. How detailed is detailed enough varies on a case-by-case basis.

4.4 Summary of Extended Hazard Analysis

This chapter introduced techniques to fill a critical gap in the ability to conduct hazard analysis on systems that exhibit the complex interactions defined in Chapter 3. Prior to this work, no known method was available to systematically and rigorously analyze system safety for collaborative control systems.

As described in Chapter 2, the common hazard analysis techniques employed in aerospace, such as FHA, FMEA, FMECA, FTA, and HAZOP, are based on *Linear Chain-of-Events* causality models, which do not consider cyclic influence. Their very foundation makes them unable to analyze collaborative relationships involving mutual influence between controllers. Most of these methods also decompose the system into individual components, which fundamentally overlooks the interactions that occur in collaboration. Finally, these techniques are hardware focused and are unable to analyze causal factors related to human and software control effectively, much less when multiple humans and automated controllers work together.

The System Theoretic model in STAMP, on which STPA is built, is able to overcome these limitations, as explained in Chapter 2. However, STPA does not specifically address eight of the nine collaborative control dynamics. This makes the technique vulnerable to omitting these interactions from consideration or to ambiguity in how to handle them. The only exception is *dynamic connectivity*, which is explored, to some extent, in the analysis of unsafe feedback paths

and unsafe control paths in STPA [50]. In addition, a past STPA extension does address *lateral coordination* [191], but it does not comprehensively explore the other collaborative dynamics.

For this reason, three extensions, collectively known as STPA-Teaming, were developed to provide a capability to analyze safety in collaborative control systems. First, the *generic collaborative control structure* provides a mechanism to express collaborative interactions in STAMP models so that they are explicitly considered in the hazard analysis.

Second, the identification of *Unsafe Combinations of Control Actions (UCCAs)* provides a systematic approach to analyzing joint control contributions from multiple controllers. The system-theoretic foundation employs abstraction to manage the combinatorial complexity as needed to remain practical for real-world systems. The method maintains the rigor of STPA as it was derived from the specification of Unsafe Control Actions (UCAs). However, the extended formulation directly covers the dynamics of *shared authority*, *dynamic authority*, and *transfer of authority*.

Finally, the extended scenario development process provides a structured method to analyze causal factors that lead to a UCCA. The approach focuses on how the interactions between the multiple controllers on the team can contribute to unsafe collective team behavior. Furthermore, it provides a mechanism to specifically consider the causal influence of the remaining six collaborative control dynamics. Finally, the process emphasizes defining scenarios at a high level and iteratively refining them to the level required to develop safety constraints. Table 4-18 summarizes how STPA-Teaming analyzes causality associated with each of the nine collaborative control dynamics.

Table 4-18. Where Collaborative Control Dynamics are Analyzed in STPA-Teaming

Collaborative Control Dynamics	Causal relationships covered by
Shared Authority	UCCA identification (Section 4.2)
Dynamic Authority	UCCA identification (Section 4.2.2)
Transfer of Authority	UCCA identification (Section 4.2.2)
Dynamic Hierarchy	Top-level scenario identification (Section 4.3.1)
Mutually Closing Control Loops	Scenario refinement (Section 4.3.2)
Cognitive Alignment	Scenario refinement (Section 4.3.2)
Lateral Coordination	Scenario refinement (Section 4.3.2)
Dynamic Membership	Scenario refinement (Section 4.3.2)
Dynamic Connectivity	Scenario refinement (Section 4.3.2)

The inherent shortfalls identified in the common hazard analysis techniques and the explanation of why and how STPA was extended in this chapter support Hypothesis 2 of this dissertation.

> *Hypothesis 2: The system-theoretic collaborative interactions framework describes component interactions that are not specifically addressed by existing hazard analysis techniques, including STPA.*

The next chapter demonstrates the application of the extended hazard analysis on a real-world system concept and evaluates the performance of the technique compared to baseline STPA. Appendices 2-4 include the case study data that was produced using the methods developed in this chapter.

Chapter 5: Case Study and Evaluation

The military is actively pursuing the development of multiple aviation concepts that aim to partner human-piloted aircraft with unmanned aircraft [28]–[31]. These novel systems typically involve a human pilot as the leader of a flight with one or more UAS acting as wingmen. Together they fly in formation and collaboratively execute complex mission tasks at the direction of the human.

These concepts are motivated by the potential cost savings and the suggested new capabilities they may provide. However, their designs face the significant challenges involved in engineering systems with team-inspired interactions described in Chapter 1.2. Multiple STPA analyses have previously explored some of the hazardous factors associated with these systems. But, as explained in Chapters 2 and 4, some of the more advanced interactions were not systematically addressed using STPA.

This chapter applies the extended hazard analysis technique developed in Chapter 4, STPA-Teaming, to one specific version of the concept: the Manned-Unmanned Teaming (MUM-T) system studied by Robertson and Hobbs [53], [213], [214]. This case study was selected for multiple reasons. The original system analyzed exhibits most of the collaborative control dynamics defined in Chapter 3. Furthermore, by relaxing some of the original assumptions, an expanded version of the system can be conceived to include all nine of the interactions. This provides an opportunity to demonstrate how STPA-Teaming systematically addresses the collaborative dynamics defined in this work.

This case study was also selected because Robertson's original analysis is the most comprehensive STPA dataset available for such a collaborative system [53]. The extensive breadth and depth of his analysis offers a good baseline of the data produced by the state-of-the-art in hazard analysis. This is important to evaluate the ability of STPA-Teaming to identify causal factors that were not otherwise considered.

Finally, this case study involves a real-world system concept that is important to the future of military aviation. Therefore, it demonstrates how STPA-Teaming can be applied to relevant systems. Furthermore, the results of the analysis can directly benefit the community to inform the development of these types of collaborative combat aircraft.

The remainder of this chapter is organized as follows. First, an overview of the MUM-T system originally and independently analyzed using STPA is provided. Next, the STPA-Teaming extensions are applied to the same system by developing the collaborative control structure, identifying unsafe combinations of control actions (UCCAs), and developing causal scenarios from the UCCAs. Then, the results of the extended and original analyses are compared. Finally, the case study is expanded to incorporate all the collaborative control dynamics and demonstrate how they are handled in the new technique.

5.1 Baseline MUM-T System & Analysis Overview

The Manned-Unmanned Teaming (MUM-T) system analyzed in the case study was originally modeled by Hobbs in STAMP to identify unsafe control actions for the system during multiple phases of flight [213]. Robertson then built off of this study to conduct an in-depth STPA analysis of the system executing a mission [53]. Those results fed a subsequent analysis of this system conducting formation flight with a single UAS [214].

The case study presented in this chapter is derived from Robertson's work [53]. Henceforth, his model of the system and its analysis are referred to as *the baseline*. The assumptions described in the baseline are initially adopted in the case study to align the foundation of the two studies so that their methods of analysis can be compared.

System Overview

An adaptation of the baseline control structure is shown in Figure 5-1. While the case study presented in this chapter considers the whole system, it focuses on the collaborative interactions between the human pilot and the UAS in the execution of shared mission tasks. This is represented by the blue region of the figure, and it is also where the causal scenario analysis in the baseline primarily focuses. In this chapter, the other controllers are abstracted away.

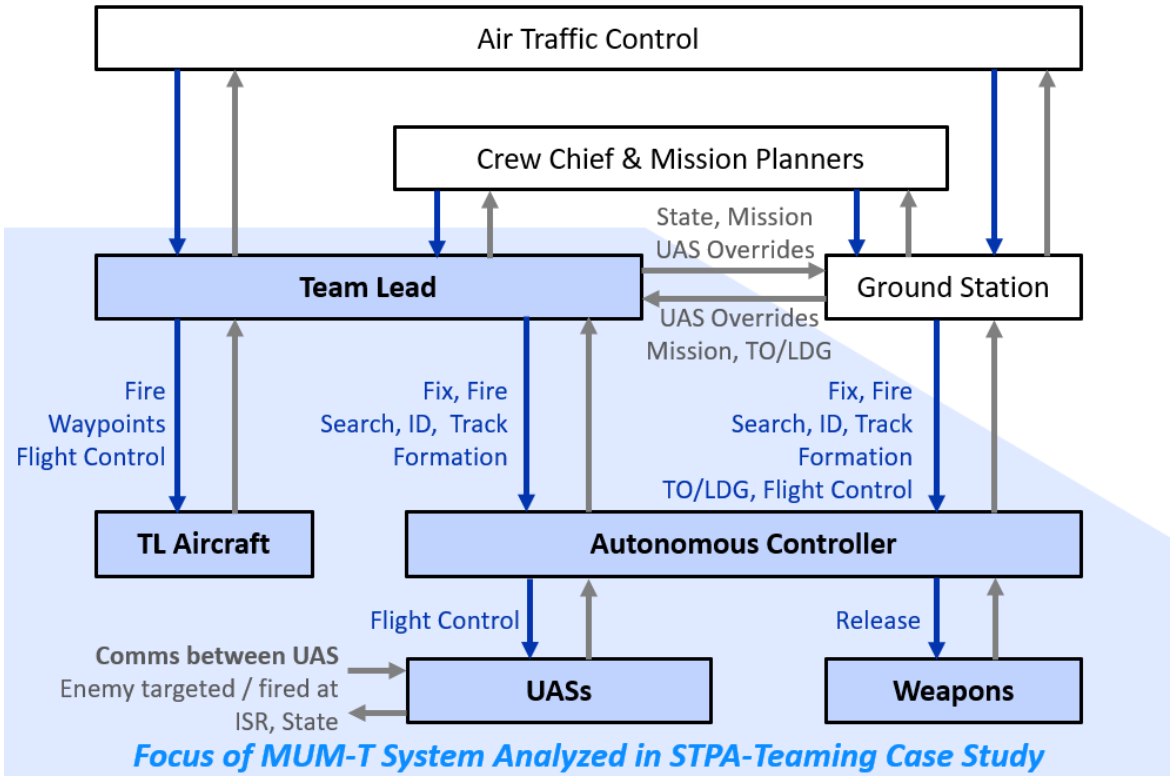


Figure 5-1. Manned-Unmanned Teaming System Control Structure (adapted from [53])

The system includes a human pilot, the Team Lead (TL), who controls her/his own aircraft and its weapon system. The TL also has authority over the Autonomous Controller that controls

the UAS airframes and their weapons. The TL is responsible for executing mission tasks, either by performing them herself/himself or by tasking UAS to do them.

For some of the tasks in the baseline, if the TL delegates them to the UAS, they must also specify which UAS is assigned to those tasks. The *fix* and *fire* commands follow this model. The *fix* command in this work represents target designation and is necessary to fire a weapon safely. The *fire* command means launching a weapon at a target. The baseline assumes that it is unsafe for multiple aircraft to *fix* on the same target or for multiple to *fire* on it.

Other tasks are delegated by the TL without designating a specific UAS. In these cases, the autonomy determines how to best allocate them to one of the UAS. The *search* for a target command is one such example, and the baseline assumes that it is not unsafe for multiple aircraft to search simultaneously. Other commands, including *track* a target and *identify (ID)* a target follow the same assumptions. For this reason, the case study abstracts the consideration for these three commands into one and applies the method to the *search* command only.

The baseline also includes a Ground Station (GS) that controls the UAS. Its primary responsibility is to launch and recover the UAS and to get them to and from the mission. However, the GS can also provide the same mission tasks as the TL (e.g., *fix*, *fire*, *search*) if the need arises. The interactions between the GS and the TL in their shared control of the UAS could also be analyzed using STPA-Teaming. However, the baseline does not explore this interaction in-depth and therefore, provides less of an opportunity to compare results. The method to analyze the TL-GS collaboration as part of the overall system hazard analysis is discussed in Chapter 6.

Finally, there are higher-level controllers in the baseline that provide control actions to the TL and the GS. These controllers, which include Air Traffic Control (ATC), the Crew Chief, and Mission Planners, are abstracted as higher authorities in the analysis presented in this chapter.

STPA Step 1 in the Baseline

The first step in hazard analysis is to identify (1) the losses that are unacceptable to the stakeholders and (2) the hazardous states of the system that could lead the system to these losses. In this case study, the losses and hazards are carried over directly from the baseline to be able to compare the information produced by the extended and the original techniques. As such, the losses and hazards for the case study are:

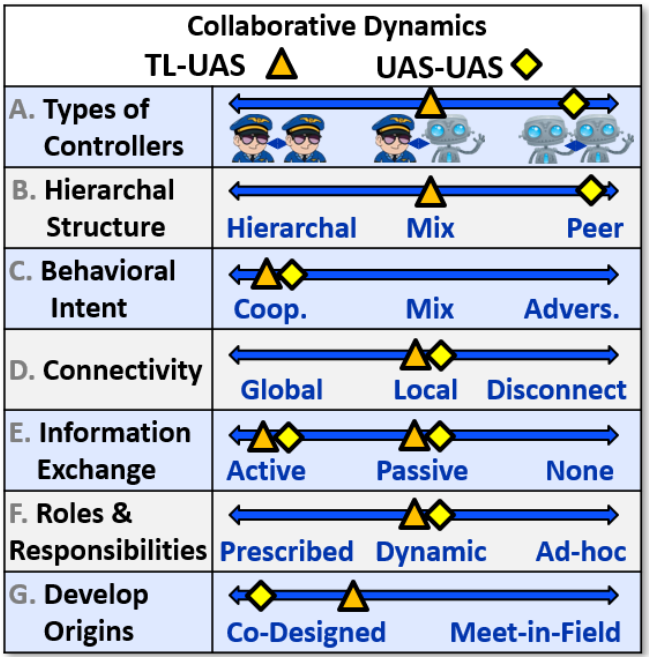
Losses [53]
L1: Death or injury of a person
L2: Destruction or damage to aircraft
L3: Non-achievement of mission
L4: Ground property damage (either military or civilian)

Hazards [53]
H1: Aircraft violate minimum separation from other aircraft or terrain [L1, L2, L3, L4]
H2: Aircraft control is lost (includes departure from stable flight) [L1, L2, L3]
H3: The system does not execute planned operations [L3]
H4: Aircraft depart approved airspace [L2, L3]
H5: The system fires at friendly forces [L1, L2, L4]

Minor modifications were made to the hazards from the baseline to emphasize the MUM-T system as a whole. The word "aircraft" is now used in plural form to recognize that the system consists of multiple aircraft. H3 replaces "aircraft" with "the system". Finally, H5 combines two hazards in the baseline originally written as: "UAV fires at friendly forces", and "Team Lead fires at friendly forces".

5.2 MUM-T Collaborative Control Structure

The second step in STPA is to model the control structure. For the case study, this involves remodeling the MUM-T system to account for its collaborative control dynamics using the conventions introduced in Chapter 4.1. The extended hazard analysis focuses on the collaboration between the human Team Lead (TL) and the UAS and those between the multiple UAS. Their interactions are first categorized using the framework developed in Chapter 3 so that the collaborative interactions can be properly reflected in the control structure (Figure 5-2).



Structure of Interactions	TL-UAS	UAS-UAS
1. Cognitive Alignment	✓	✓
2. Lateral Coordination	✓	✓
3. Mutually Closed-Loop	✓	✓
4. Shared Authority	✓	✓
5. Transfer of Authority	✓	✓
6. Dynamic Authority	✓	✓
7. Dynamic Hierarchy	✗	✗
8. Dynamic Membership	✓	✓
9. Dynamic Connectivity	✓	✓

Figure 5-2. Categorization of the Human-Machine & Multi-Machine Interactions in MUM-T

The types of controllers [A] involve both human-machine and multi-machine interactions. The hierarchal structure between the human and the machines has a mix of both supervisory control and peer interactions [B]. In MUM-T, the human tasks the UAS using control, but s/he also coordinates with UAS as a peer, for instance, when the UAS provides a fix on a target for the TL to fire on. Conversely, the UAS interact exclusively as peers.

The next four dimensions are common between the two relationships. All controllers are cooperative in intent [C]. Connectivity is local only [D], as there are no guarantees that every member of the team can communicate with everyone else. Information exchange includes active messaging, but controllers may also observe the behaviors of their teammates [E]. Roles and

responsibilities are dynamic as the team must determine which controller will close which control loop during execution [F]. Finally, the developmental origins may vary if the UAS are co-designed, but the human pilot is originally trained to operate with human wingmen [G].

Given the baseline assumptions, the MUM-T system exhibits up to eight of the nine collaborative control dynamics defined in Chapter 3. As previously mentioned, all controllers may laterally coordinate with each other, especially to execute coupled tasks such as *fix* and *fire*. Similarly, the controllers exercise team cognition by relying on shared and distributed information to synchronize their models and decisions.

The controllers can mutually close each other's control loops when one provides a fix for another to fire. There are a number of mechanisms employed in the real world to accomplish target designation. In some cases, the designating controller uses a sensor, such as a camera, to observe information about the target and passes that feedback to the firing controller. However, in other situations, the designator illuminates the target for the firer to sense directly, as is the case in laser designation or bistatic radar returns. To maintain generality, both paradigms are considered.

The MUM-T controllers share authority over the joint mission process. There is dynamic authority as they must determine task allocation during execution. Similarly, transfer of authority occurs as the system may hand off the responsibility to issue certain control actions between controllers on the team. The membership of the UAS is dynamic as they can be added or removed from the team. Finally, the network topology is expected to vary leading to dynamic connectivity.

The only collaborative control dynamic not exhibited given the baseline assumptions is dynamic hierarchy. Therefore, the case study excludes this type of interaction in the main analysis (Sections 5.2-5.5) so that the assumptions are consistent with the baseline for the purpose of comparing the results. However, the MUM-T assumptions are later relaxed in Section 5.6 to introduce dynamic hierarchy into the system and demonstrate how STPA-Teaming handles this interaction.

While remaining consistent with the baseline assumptions, the MUM-T system is remodeled according to the generic collaborative control structure conventions introduced in Chapter 4.1 (Figure 5-3). The control structure now includes the shared process controlled by both the TL and the UAS. The control lines from the TL to the UAS also include non-control items, such as coordination messages and observations. The two UAS shown represent a variable number of such interchangeable controllers.

The baseline does not include the ability for the TL to provide a *fix* using her/his own aircraft. As such, the TL does not have this ability in the present case study. Similarly, because the TL can navigate her/his aircraft, the following analysis assumes the TL can navigate to *search* for a target.

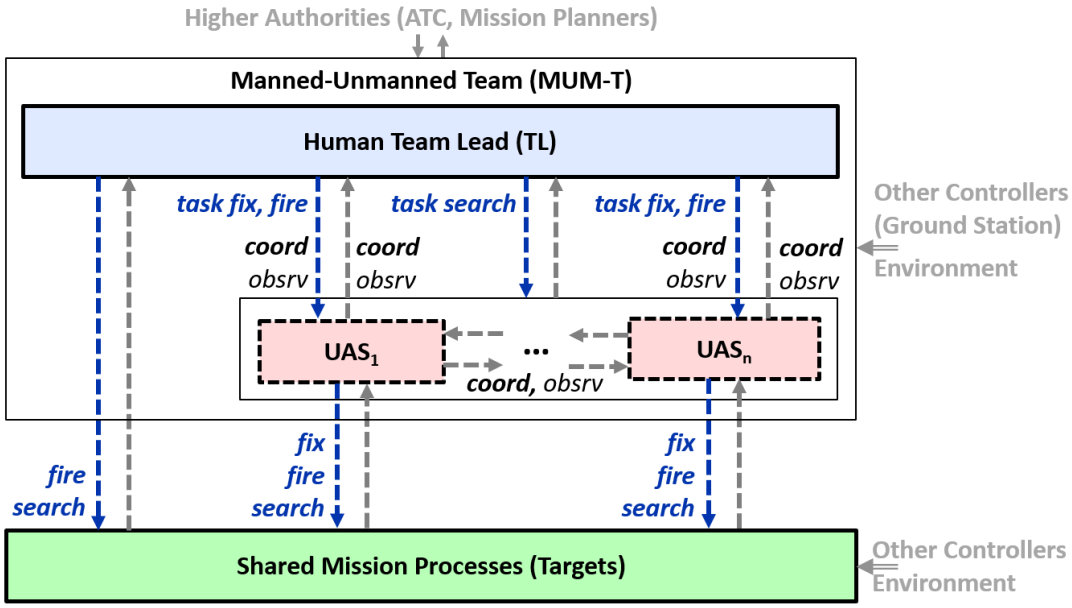


Figure 5-3. Collaborative Control Structure for the MUM-T System

5.3 MUM-T Unsafe Combinations of Control Actions

The third step of STPA identifies the Unsafe Control Actions (UCAs) for the commands specified in the control structure. The extended hazard analysis mirrors this step by finding the Unsafe Combinations of Control Actions (UCCAs) for the commands to the shared process.

In the MUM-T case study, the control output consists of various combinations of *fix*, *fire*, and *search* commands provided by the different controllers on the team. The UCCA Identification Algorithm from Chapter 4.2 is executed using the tool prototyped in MATLAB, which supports the analyst by automating some of the steps of the process.

Inputs

The following three inputs are specified from the MUM-T system for the algorithm. First, a tuple encodes the control actions that each of the three controllers can contribute to the shared process. Second, UAS_1 and UAS_n are designated as interchangeable controllers, as they are of the same type and at the same level of hierarchy.

The third input is a special interaction to influence the refinement of UCCAs with the *fix* command. The baseline states that it is unsafe for multiple controllers to fix a target simultaneously. A possible reason for this is mutual interference, whether in the spatial or in the radio frequency (RF) domain. The resulting effect of multiple controllers providing a fix can be equivalent to no fix provided collectively. Therefore, a heuristic is created to include instances of multiple controllers providing a fix in the refinement of UCCAs that involve no collective fix being provided.

Abstracted UCCAs

The automation first creates four tables that enumerate all the potential abstracted UCCAs using the formulation from Table 4-14. The tables cover Type 1-2 UCCAs (provide/not provide) and Type 3-4 UCCAs (start/end before/after another), each using Abstraction 2a (team issues combinations of control actions) and Abstraction 2b (combinations of controllers issue common control action).

Table 5-1 presents the full automated output for the MUM-T Abstraction 2a Type 1-2 UCCAs. The human analyst then reviews the table and specifies the context, if one exists, in which each combination is unsafe. If a combination has multiple contexts, the analyst can add new lines as necessary and track multiple instances using the subindex (*sid*). In the table, *items italicized and in green font are human inputs*.

The analyst also designates the control actions that are *relevant* in the context. This is accomplished by listing the identifier of the control actions from the third column, where *1 = fix, 2 = fire,* and *3 = search*. In cases where duplicate combinations are found, with the same context and the same relevant control actions, they are tracked using the *same* column and excluded from the remainder of the analysis. In Table 5-1, six unique UCCAs are identified: items 1-4 and 10-11.

Table 5-1. MUM-T Abstraction 2a Type 1-2 UCCAs (Green: Human Inputs)

id	team	team	sid	same	context	relevant*
1	¬fix	¬{fire,search}	*1*	0	*when there are mission tasks to execute [H3]*	*2,3*
2	fix	¬{fire,search}	*2*	0	*when target will compromise mission if fixed but not fired on [H3]*	*2*
3	¬fix	{fire,search}	*3*	0	*when target fired-on must be fixed [H3, H5]*	*2*
4	fix	{fire,search}	*4*	0	*when fixed target is different than target fired-on [H3, H5]*	*2*
5	¬fire	¬{fix,search}	0	*1*	*when there are mission tasks to execute [H3]*	*2,3*
6	fire	¬{fix,search}	0	*3*	*when target fired-on must be fixed [H3, H5]*	*2*
7	¬fire	{fix,search}	0	*2*	*when target will compromise mission if fixed but not fired on [H3]*	*2*
8	fire	{fix,search}	0	*4*	*when fixed target is different than target fired-on [H3, H5]*	*2*
9	¬search	¬{fix,fire}	0	*1*	*when there are mission tasks to execute [H3]*	*2,3*
10	search	¬{fix,fire}	*5*	0	*when engaging a known target is higher priority than searching [H3]*	*1,2*
11	¬search	{fix,fire}	*6*	0	*when searching for another target has higher priority [H3]*	*1,2*
12	search	{fix,fire}	0	0		

** Control actions relevant to the UCCA in column 3. 1 = fix, 2 = fire, 3 = search*

Table 5-2, Table 5-3, and Table 5-4 list the unique abstracted UCCAs identified for Abstraction 2a Type 3-4, Abstraction 2b Type 1-2, and Abstraction 2b Type 3-4 respectively. The full set of combinations produced by the automation in each case is included in Appendix 2. The items in these three tables, along the six items found in Table 5-1, form the set of 19 abstracted UCCAs carried forward in the analysis.

Table 5-2. MUM-T Abstraction 2a Type 3-4 UCCAs (Green: Human Inputs)

id	Team ...	then ...	sid	context	relevant*
15	E(fix)	S(fire,search)	*1*	*when target fired on must be fixed [H3, H5]*	*2*
17	S(fire)	S(fix,search)	*2*	*when target fired on must be fixed [H3, H5]*	*1*

** Control actions relevant to the UCCA in column 3. 1 = fix, 2 = fire, 3 = search*

Table 5-3. MUM-T Abstraction 2b Type 1-2 UCCAs (Green: Human Inputs)

id	Ci	Cj(s)	sid	context
37	¬fix	¬fix	*1*	*when there is a priority target to engage & a teammate can fire [H3]*
38	fix	¬fix	*2*	*when tasked entity is not capable and another is [H3]*
39	fix	fix	*3*	*when that creates mutual interference [H1, H3]*
40	¬fire	¬fire	*4*	*when there is a priority target to engage and a teammate can fix [H3]*
41	fire	¬fire	*5*	*when tasked entity is not capable and another is [H3]*
42	fire	fire	*6*	*when that creates excessive effects [H3, H5]*
43	¬search	¬search	*7*	*when no targets have been found [H3]*
44	search	¬search	*8*	*when tasked entity is the only one capable for higher priority task and teammate can search [H3]*

Table 5-4. MUM-T Abstraction 2b Type 3-4 UCCAs (Green: Human Inputs)

id	Ci ...	then Cj ...	sid	context
47	S(fix)	E(fix)	*1*	*when that creates mutual interference [H1, H3]*
48	E(fix)	S(fix)	*2*	*when that creates a large gap in a fix handoff [H3]*
56	E(search)	S(search)	*3*	*when that creates an excessive gap in the search [H3]*

Refined UCCAs

The abstracted UCCAs identified in the four tables above are then refined by automation. The tool determines all the ways the different controllers can contribute to the collective output of each UCCA using Equation (17) and the discussion in Section 4.2.4. It then prunes combinations that are duplicated across the interchangeable controllers UAS_1 and UAS_n with Equation (21).

Finally, the algorithm prioritizes the remaining refined UCCAs and sorts the output by decreasing priority. The following prioritization heuristic is implemented for this case study. Instances in which a single controller provides all the unsafe control actions are assigned the lowest priority (priority 4), as they do not involve collaborative control.

Next, for Abstraction 2a, which explores combinations of different control actions, cases that include two or more instances of the same control action are downgraded to priority 3. These items are repetitive over those that just include one instance of that action. An exception is made based on the special interaction for the *fix* command, and instances of two fixes are given priority

2. For Abstraction 2b, which specifically examines multiple controllers providing the same action, instances of three or more of the same action are given priority 3. All other cases are labeled with the highest priority value of 1.

Table 5-5 shows the refined, pruned, and prioritized output of UCCA 10 initially identified in Table 5-1 as an example for Type 1-2 UCCAs. The same context specified for the abstracted UCCA applies to all of its refined children, which is not shown in Table 5-5 due to lack of space. Similarly, Table 5-6 refines UCCA 15 from above to exemplify the output for Type 3-4 UCCAs.

Table 5-5. MUM-T Refinement of UCCA 10 (Example for Type 1-2 UCCA)

id	TL		UAS1			UASn			priority
10.5.1	¬fire	search	¬fix	¬fire	¬search	¬fix	¬fire	¬search	1
10.5.2	¬fire	¬search	¬fix	¬fire	search	¬fix	¬fire	¬search	1
10.5.4	¬fire	search	fix	¬fire	¬search	fix	¬fire	¬search	2
10.5.3	¬fire	search	¬fix	¬fire	search	¬fix	¬fire	¬search	3
10.5.5	¬fire	¬search	¬fix	¬fire	search	¬fix	¬fire	search	3
10.5.6	¬fire	search	¬fix	¬fire	search	¬fix	¬fire	search	3

Table 5-6. MUM-T Refinement of UCCA 15 (Example for Type 3-4 UCCA)

id	TL	UAS1		UAS2	priority	Context
15.1.1	**F** S(fire)	E(fix)			1	*when target fired on must be fixed [H3, H5]*
15.1.3		E(fix)		**F** S(fire)	1	
15.1.2		E(fix)	**F** S(fire)		4	

Overall, the tool generates 63 refined, pruned, and prioritized UCCAs from the original list of 19 abstracted UCCAs. In this case study only the 35 UCCAs of priority 1 and 2 are evaluated in scenario development. Lower priority items are still available and could be analyzed in future work to assess how much additional information they contribute to identifying causal factors.

5.4 MUM-T Causal Scenarios

In the fourth step of STPA, loss scenarios are developed to identify the causal factors that contribute to unsafe control. Using STPA-Teaming, scenarios are identified to explain how the MUM-T UCCAs defined above could occur. The process, as described in Chapter 4.3, first defines top-level scenarios to reason about control actions internal to the team. It then iteratively refines those scenarios using the template shown in Figure 4-21.

The notation informs the part of the process being executed. For example, "S-37.1.X" indicates the scenario is traced to UCCA 37.1 and its refined scenarios. The last digit designates the top-level scenario called in Step 1 (see Table 4-16), so S-37.1.1 is for top-level scenario #1. The refinement specifies the causal factors considered in internal control (Step 2) and collaborative control (Step 3). Because there are eight different top-level scenarios specified (recall Table 4-16 and Table 4-17), the other factors explored in Step 4 are included under S-37.1.9.

Two examples follow to collectively illustrate the whole process. The first demonstrates the general process to develop scenarios on a Type 1-2 UCCA using the framework detailed in Figure 4-21. All the top-level scenarios are defined, and then two of them, #1 and #3, are iteratively refined to show how a subset of the causal factors are identified. The second example demonstrates how to define the top-level scenarios for a Type 3-4 UCCA. Appendix 3 contains the full dataset of the scenarios identified in the case study.

Example 1. Scenario Development for Type 1-2 UCCA

UCCA 37.1 (abstracted UCCA): Controller C_i does not fix and no other C_j fixes when there is a priority target to engage and a teammate able to fire [H3]

> **UCCA 37.1.1 (refined UCCA)**: UAS_1 does not fix and UAS_n does not fix when there is a priority target to engage and a teammate able to fire [H3] (refined in Appendix 2)

Figure 5-4 shows the four relevant internal control combinations that can lead to the collective output of the UCCA. It enumerates whether or not the Team Lead (TL) tasks either UAS_1 or UAS_n to fix. The figure is only provided here to demonstrate how the internal control combinations are captured into the top-level scenarios, as listed below each case in the figure. It is not itself necessary to execute the analysis. Because UAS_1 and UAS_n are interchangeable, case 1c in the figure is a duplication of 1b.

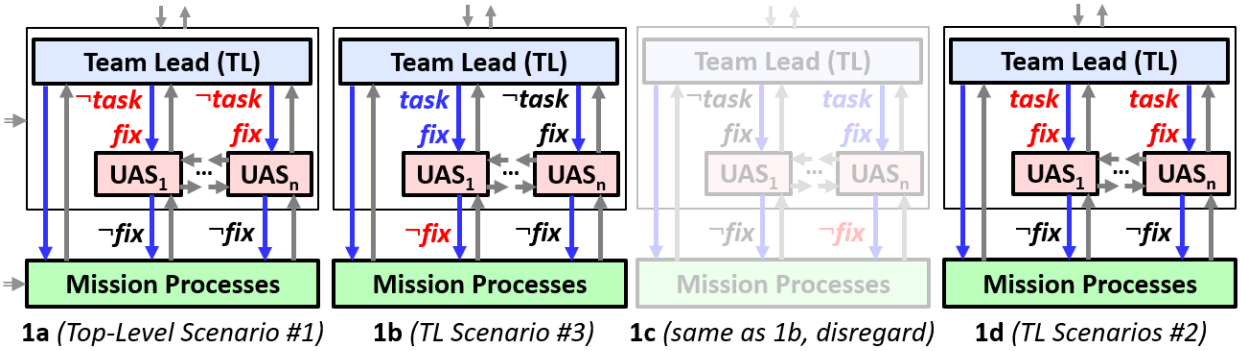


Figure 5-4. Internal Control Combinations that Result in No Controller Providing a Fix

S-37.1.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to provide safe collective control. Here, the TL does not task any UAS to fix.

- **(Step 2: Internal Control)** ***Unsafe Control Input:*** TL misinterprets direction from higher authorities that the team should not fix any targets. ***Refinement:***
  - TL previously received a command not to provide a fix. However, that command is now outdated, but the TL does not receive the updates.
  - *More listed in Appendix 3…*

- **(Step 2: Internal Control)** ***Inadequate Process Model:*** TL has one or more of the following inadequate process model variables: (1) TL does not believe there is a target to engage, (2) TL does not believe there is a teammate able to fire (to couple with the fix), (3) TL believes that a UAS has already been tasked to fix, or (4) TL does not believe there is a teammate available to task the fix. ***Refinement:***

- *Incorrect feedback.* The interface displays stale feedback of the team state, which shows one of the UAS as tasked to fix from a now outdated tasking.
- *More listed in Appendix 3 (Incorrectly Interpreted Feedback, Feedback not sent) …*

- **(Step 2: Internal Control)** ***Inadequate Control Algorithm:*** TL has accurate information about the target and the team capabilities, but still chooses not to task a UAS to issue the fix. ***Refinement:***
  - TL is currently busy and prioritizing other operating tasks but intends to task a UAS soon. However, s/he later forgets to do this due high workload.
  - *More listed in Appendix 3…*

- **(Step 2: Internal Control)** ***Unsafe Control Path:*** TL intends to task a UAS but is unable to do so. ***Refinement:***
  - The communication channel to do so is currently inadequate. This could be due to RF jamming, communications fading, misconfigured encryption settings, or other reasons.
  - *More listed in Appendix 3…*

- **(Step 3: Collaborative Dynamic)** ***Mutually Closing Control Loops*****:** *Not applicable. The top-level scenario focuses on the TL decision not to task the fix. This internal control loop is not closed through any other controller.*

- **(Step 3: Collaborative Dynamic)** ***Cognitive Alignment*****:** *Not applicable. The top-level scenario focuses on the TL decision not to task the fix. No other controller is involved in this decision. See Inadequate Process Model, Inadequate Control Algorithm instead.*

- **(Step 3: Collaborative Dynamic)** ***Dynamic Membership:***
  - TL intends to task a UAS; however, the set of UAS participating on the team fluctuates too much at the moment for the TL to have sufficient confidence that the assigned UAS will still be in the team to carry out the task. ***Refinement:***
    - The Ground Station is swapping out UAS that need to be refueled with new ones.
  - *More listed in Appendix 3…*

- **(Step 2: Internal Control)** ***Dynamic Connectivity:***
  - TL tasks a UAS to fix, but due to the dynamic topology, the tasking is not adequately routed to the intended UAS(s). This could occur because the message is degraded over too many hops or because the changing topology does not support information routing requirements (e.g., path, timing, …).
  - *More listed in Appendix 3…*

S-37.1.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, (1) the TL tasks multiple UAS to fix and, therefore, none execute it, or (2) the TL tasks a UAS to fix that cannot fix. *See Appendix 3 for refinement…*

S-37.1.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks a single capable UAS to fix, but the UAS does not fix.

- **(Step 2: Internal Control)** ***Unsafe Control Input:***
  - The UAS tasked to fix is overridden by another controller (e.g., the Ground Station, a different Team Lead, or a cyber attacker) to perform a different action.

- **(Step 2: Internal Control)** ***Inadequate Process Model:*** The tasked controller has the following inadequate process model variable: (1) it does not know how to provide the fix command, (2) it incorrectly believes the fix command has already been provided.
  - *Inadequate Feedback:* The controller tasked to fix has inadequate sensor feedback of the target to execute the fix.

- **(Step 2: Internal Control)** ***Inadequate Control Algorithm:*** *addressed in cognitive alignment.*

- **(Step 2: Internal Control)** ***Unsafe Control Path:*** the controller tasked to fix intends to provide its assigned command. However, it does not deliver the command to the controlled process. ***Refinement***:
  - The targeting equipment malfunctions, and it is unable to do so.

- **(Step 3: Collaborative Dynamic)** ***Mutually Closing Control Loops:*** In this system, if the controller that provides the fix is different from the controller that fires, the two controllers collaborate to fix and fire on the target. Their two control loops can be coupled as represented in Figure 5-5. ***Refinement:***

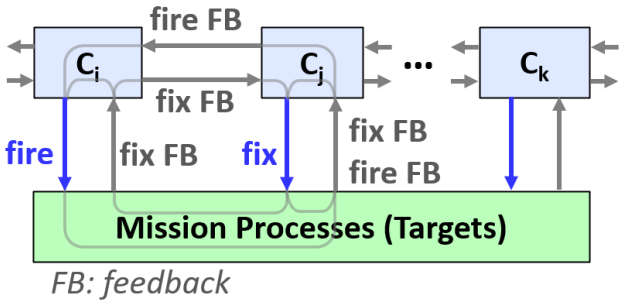


Figure 5-5. Coupled Control Loops for Controllers Fixing and Firing on a Target

  - ***Feedback about Controlled Process from Collaborators*****:** The feedback the UAS tasked to fix receives from collaborators leads it to believe it does not need to provide the fix command. ***Refinement***:
    - The UAS tasked to fix (e.g., UAS_1), prior to starting the fix, receives incorrect feedback from a teammate that the current fix is adequate to fire. Based on this feedback UAS_1 does not provide the fix. ***Refinement:***
      - The teammate broadcasts feedback for a different fix, of a different target, provided by a different teammate.
      - (Human-Machine) UAS_1 provides the fix for the TL (human) to fire. The TL inadvertently labels feedback for UAS_1's fix as adequate. This may occur because the TL is under heavy workload and makes a mistake. It could also occur because there is a human-machine semantic mismatch in the feedback.
    - *More listed in Appendix 3…*

- ***Feedback about Collaborator Control Actions from Controlled Process*:** The feedback the UAS tasked to fix receives from the process leads it to believe it does not need to provide the fix command. ***Refinement:***
  - The UAS tasked to fix a target (e.g., UAS_1) temporarily receives fix energy off the target provided by another UAS (e.g., UAS_2). This leads UAS_1 to drop the task in the belief that UAS_2 is executing it. However, this may have been a stray command from UAS_2 (e.g., in its process to fix a different target, temporary system malfunction, …). As a result, no UAS provides the fix command as tasked by the TL.

  ***Note**: The UCCA analyzed does not explicitly show the fire command. If this control action coupling was not considered in this step, it would be explicitly covered in UCCA 3.3, in which the Team provides the fire command and does not provide the fix command.*

- **(Step 3: Collaborative Dynamic)** ***Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. Figure 5-6 illustrates a possible misalignment of hypothetical model elements between MUM-T controllers to conceptualize this dynamic for analysis. The figure is derived from Figure 4-6 and items highlighted in yellow are inconsistent between the two controllers. ***Refinement:***

HUMAN - TL Mental Model (example)			
Controlled Process	Fix+Fire: Tgt A, Time B, Procedure C		
Collaborators			
Set of Controllers	TL	UAV1	UAV2
States	$\{xyz...\}_{\sim t}$	$\{xyz...\}_{\sim t-1}$	$\{xyz...\}_{\sim t}$
Responsibilities	Task UAVs Fire	Fix	-none-
Connectivity	N/A	Direct	Indirect
Info Req (to)	N/A	Task Coord	Tasking
Info Req (from)	N/A	Task Coord	Status
Confidence	High	High	Low
Environment	…		
Other Controllers	…		

MACHINE - UAV1 Process Model (example)			
Controlled Process	Fix+Fire: Tgt A, Time B, Algorithm D		
Collaborators			
Members	TL	UAV1	UAV2
States	$\{xyz...\}_{t}$	$\{xyz...\}_{t}$	$\{xyz...\}_{t}$
Responsibilities	Task UAVs -none-	Fix	Fix
Connectivity	Direct	N/A	Direct
Info sent to	Status	N/A	Task Coord
Info received from	Tasking	N/A	Task Coord
Confidence	High	Low	High
Environment	…		
Other Controllers	…		

Figure 5-6. Misalignment of Hypothetical Models between MUM-T Controllers

- ***Construction***: The process models and/or control algorithms are not adequately or consistently built across the team to support collaborative control. ***Refinement***:
  - (Human-Machine) The control algorithms on the UAS are not compatible with how the TL specifies the fix task. This prevents the UAS from accepting the task or being able to collaborate with another controller in a coupled fire task. This could occur due to configuration management problems with the UAS or the TL interface software, variation in how the TL was trained to work with the UAS, and conflicting past experience the TL has working with human wingmen instead.
  - *More listed in Appendix 3…*

- ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
  - The different UAS receive different versions of the <fix, fire> task specification by the TL. This could occur because the TL periodically updates how the task is specified, given slight changes in her/his model of the mission. The different UAS may receive the task specification across different replan cycles. As a result, the controllers on the team do not have sufficient common knowledge of the task to coordinate in coupled fix & fire tasks.
  - The different UAS have different beliefs about how many and which UAS are participating on the team. This may occur due to dynamic membership, with UAS cycling on and offline, and dynamic connectivity that prevents timely distribution of this information. This could result in the UAS assigned to fix (e.g., UAS_1) not believing that the UAS assigned to fire (e.g., UAS_2) is currently participating in the team. As such, UAS_1 does not believe conditions are satisfied for it to provide a fix.
  - *More listed in Appendix 3…*
- ***Model Updates***: Some elements of some of the process models are not adequately or consistently updated across the team. ***Refinement***:
  - ***Vertical Coordination (Control)***:
    - The TL overrides one of the UAS with a command inconsistent with the laterally coordinated task execution. For example, UAS_1 (tasked to fix) and UAS_2 (tasked to fire) have laterally coordinated the details of the coupled fix and fire tasks. However, the TL accelerates the UAS_2 timeline to make it available for other tasks but is not aware that this now conflicts with the coordinated plan.
    - *More listed in Appendix 3…*
  - ***Lateral Coordination (Communication):***
    - (Human-Machine) Lateral coordination between UAS_1 (tasked to fix) and TL (self-tasked to fire) is hindered by human-machine asymmetry in information semantics. Both controllers interpret shared information differently. The TL encodes subtleties and ambiguities, which are common for humans to handle, but difficult for machines to process precisely. The UAS is unable to describe task parameters that exceed its programmed bounds but may be necessary to overcome unforeseen issues. This hinders the execution of the coupled fix-fire commands.
    - (Human-Machine) Lateral coordination between UAS_1 (tasked to fix) and TL (self-tasked to fire) is hindered by human-machine asymmetry in information timing. For example, the machine provides excessive information requests, which interrupt the flow of the TL in the execution of the coupled fix-fire commands.
    - *More listed in Appendix 3…*

        - ***Lateral Coordination (Observation)***: The controller selected to execute the fix task observes another controller maneuvering in a way that is consistent with executing that task and incorrectly believes it will provide it. The maneuvering controller is not aware of this misinterpretation. As a result, the assigned controller does not provide the fix.
        - ***Prediction***: UAS_1 (tasked to fix) has a model of the coupled fix & fire task that expects to receive certain coordination messages and observations by certain milestones. If the other controller <TL, UAS_2> (tasked to fire) is delayed or takes an unexpected trajectory to fire, UAS_1 may lose confidence in its ability to fulfill the task and, therefore, incorrectly decide not to provide the fix task either.
        - ***Other Information Sources***: UAS1 (tasked to fix) observes a change in the environment (e.g., change of weather) that leads it to believe that the other controller <TL, UAS_2> (tasked to fire) will not be able to support the fire command.
    - ***Decision-Making***: The process controllers use to decide what control and communications actions they provide are inadequate or inconsistent across the team. ***Refinement***:
        - Despite adequate communication channels, the distributed decision-making process is too slow to keep up with the dynamic state of the shared-controlled process and does not converge fast enough on a solution. ***Refinement:***
            - Two controllers attempt to coordinate the coupling of the fix and fire commands. However, at the conclusion of every iteration of distributed planning, the state of the system has changed enough that the plan is no longer relevant. As a result, the team is unable to reach a valid plan.
        - *More listed in Appendix 3…*
    - ***Capacity***: The capacity of one of the controllers is inadequate to enable effective alignment of team cognition. ***Refinement***:
        - One of the controllers, TL or UAS, has a runaway internal process that is using up processing resources (e.g., system failure, denial of service). As a result, the controller is unable to keep up with the coordination demands of the rest of the team, which prevents the whole team from reaching consensus or achieving execution goals.

- **(Step 3: Collaborative Dynamic)** ***Dynamic Membership:***
    - The loss of a teammate makes the UAS tasked to fix drop its task. For example, at the time the UAS was tasked to fix (e.g., UAS_1), another controller was tasked to fire collaboratively with it (e.g., UAS_2). UAS_2 then departs the team (e.g., taken offline by another controller, lost in mission, …), and its fire task gets reassigned. However, the retasking is not known to UAS_1, which therefore does not know whom to coordinate with to couple its fix task.

- *More listed in Appendix 3…*

- **(Step 3: Collaborative Dynamic)** ***Dynamic Connectivity:***
  - The process of providing the fix command leads UAS_1 to temporarily disconnect from the team (e.g., physically traveling to the location to fix causes it to disconnect). Because connectivity is required to coordinate the coupled fix and fire tasks with another controller, UAS_1 drops its task. However, it also does not reclaim the task when connectivity is regained.
  - *More listed in Appendix 3…*

S-37.1.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyway, which leads to unsafe collective control.

> ***Note***: *The context of this UCCA does not apply to this high-level scenario (see Figure 5-4). The UCCA assumes that it is necessary to provide tasks.*

S-37.1.5 (Step 1: Top-Level Scenarios #5): The TL control actions to the shared process are unsafe in combination with how it directs the UAS.

> ***Note***: *The context of this UCCA does not apply to this high-level scenario (see Figure 5-4). The TL does not have the ability to provide the fix command, as specified in the system assumptions.*

S-37.1.9 (Step 4: Other Factors): The controllers on the team do not provide any commands due to other factors beyond the interactions between the controllers on the team.

- *Unsafe Feedback Path:* see *Inadequate Process Model* in S-37.1.3
- *Unsafe Control Path:* see *Unsafe Control Path* in S-37.1.3
- *Unsafe Process Behavior:* the UAS provides a fix, but the target deploys countermeasures or performs defensive maneuvers that overcome its effects.

As shown in the example, there can be some overlap between the top-level scenarios. For example, the unsafe feedback path and unsafe control path factors here were found both in S-37.1.3 and S-37.1.9. As stated in Chapter 4.3, the analytical process favors overlap instead of gaps in consideration. Because the purpose is to identify causal factors to eliminate or mitigate them, it is acceptable to reiterate a factor, even if it reduces the analytical efficiency.

Example 2. Scenario Development for Type 3-4 UCCA (Top-Level Scenarios Only)

The second example demonstrates how the top-level scenarios defined in Table 4-17 are created for a Type 3-4 UCCA. The refinement of the top-level scenarios follows the same process as in the first example. So, they are excluded here for brevity but are available in Appendix 3.

UCCA 56.3: Controller C_j ends providing the search command before C_i starts providing the search command when that creates an excessive gap in a search handoff [H3]

> UCCA 56.3.1: TL ends search before UAS_1 starts search when (*same context as above*)
>
> UCCA 56.3.2: UAS_1 ends search before TL starts search when (*same context as above*)
>
> UCCA 56.3.3: UAS_n ends search before UAS_1 starts search when (*same context as above*)

Figure 5-7 shows the two relevant internal control combinations for each of the collective outputs in the three refined UCCAs. UCCA 56.3.1 is covered by cases 1a and 1b, UCCA 56.3.2 by 2a and 2b, and UCCA 56.3.3 by 3a and 3b. As in the previous example, the figure is not necessary for the process, but it is included here to show how the possible internal control are covered by the top-level scenarios.

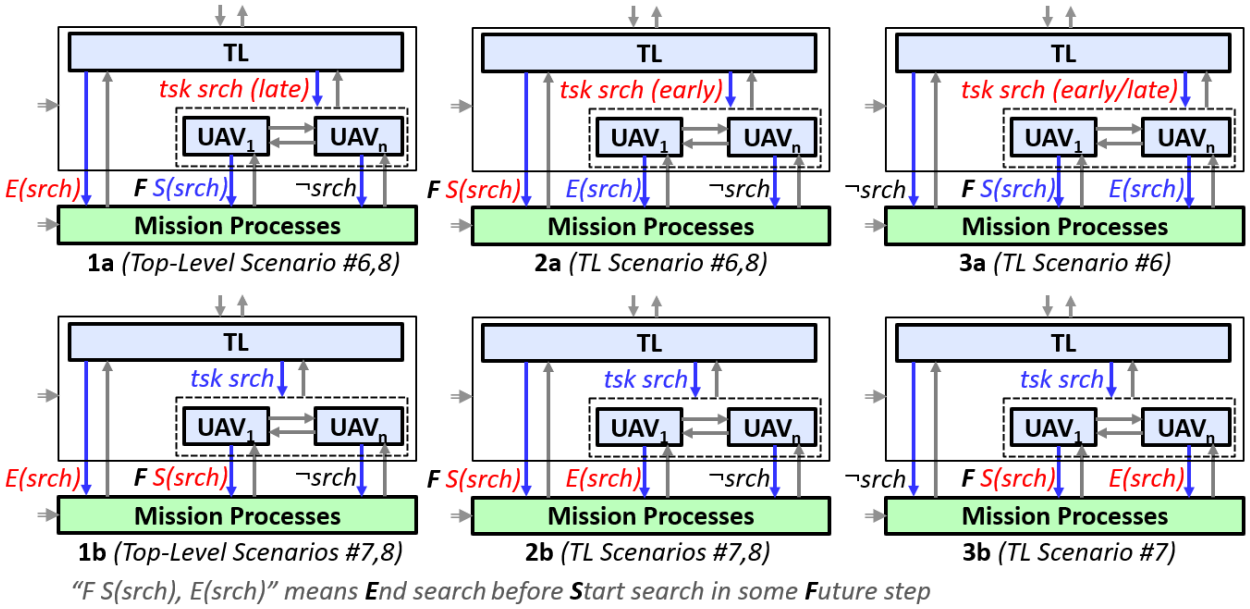


Figure 5-7. Internal Control Combinations that Result in an Unsafe Gap in a Search Handoff

S-56.3.6 (Step 1: Top-Level Scenarios #6): TL directs the UAS in a way that leads to unsafe temporal sequencing. Here, the TL tasks the UAS to start or end the search in a way that creates a gap with the TL's planned start or end to the search.

***Note**: By system definition, the TL specifies a general search task for the UAS team, and the automation is responsible to assign UAS(s) to fulfill the task. This implies that the autonomy is responsible to manage the specification of handoffs between two UAS.*

- ***Refinement** in Appendix 3…*

S-56.3.7 (Step 1: Top-Level Scenarios #7): TL adequately directs the UAS, but the way in which the UAS execute directions leads to unsafe temporal sequencing. Here, a UAS starts or stops the search in a way that creates a gap (1) with the TL start or stop of the search or (2) with the start or stop from other UAS.

- ***Refinement** in Appendix 3…*

S-56.3.8 (Step 1: Top-Level Scenarios #8): TL provides control actions to the controlled process that are unsafe in temporal sequencing with how it directs the UAS. Here, the TL starts or ends the search in a way that creates a gap with how the TL tasked the UAS.

- ***Refinement** in Appendix 3…*

Heuristics Employed in Scenario Development for this Case Study

Two heuristics are applied to the development of scenarios for this case study. First, the scenarios are developed in a non-linear sequence. This means each scenario is initially developed to address the specific UCCA listed above it, as shown in Example 1. Then, as additional UCCAs are found to have similar causal factors, the original scenario is modified to integrate the lower-level variations. The scenarios for the later UCCAs are then traced to the original scenarios instead of repeating the same information multiple times.

As an example, an excerpt of Scenario S-37.1.1 for UCCA 37.1, "C_i does not fix and no other C_j fixes when …" (see Example 1 above) is originally written as:

> **Initial Version - S-37.1.1 *Unsafe Control Input:*** TL misinterprets direction from higher authorities that the team should not fix any targets.

Several UCCAs subsequently analyzed have similar control combinations, such as UCCA 40.4 "C_i does not fire and no other C_j fires when …" and UCCA 43.7 "C_i does not search and no other C_j searches when …". To avoid repetition, the excerpt of S-37.1.1 above is modified to the form below, which covers the later UCCAs. Then, the matching scenario headers under S-40.4.1 and S-48.7.1 are traced to S-37.1.1. The following excerpt of S-37.1.1, which is not traced to any other scenario, is then labeled as a *unique scenario* in the data analysis performed in Section 5.5.

> **Modified Version - S-37.1.1 *Unsafe Control Input:*** TL misinterprets direction from higher authorities that the team should not <fix, fire on, search for> any targets.
>
> **S-40.4.1 *Unsafe Control Input:*** see S-37.1.1.
>
> **S-48.7.1 *Unsafe Control Input:*** see S-37.1.1.

The non-linear sequence consolidates multiple similar scenarios together to reduce the duplication of information. The aim is to make the analysis more readable and efficient to derive safety constraints from it.

The second heuristic focuses the search by analyzing Abstraction 2b UCCAs (combination of controllers issuing a common control action) before Abstraction 2a UCCAs (combinations of different control actions provided by the team). For example, by first analyzing UCCA 37.1, "C_i does not fix and C_j does not fix when …" (see Example 1 above), numerous causal factors are found that are relevant to UCCA 3.3 "the team provides the fire command but does not provide the fix command when…". UCCA 3.3 can leverage the analysis of UCCA 37.1 and focus more on why no controller would provide the fix if a controller fires and vice versa.

5.5 Case Study Comparison of Results

The results of the MUM-T case study presented above are compared to those produced by STPA in the *baseline* analysis [53]. The objective of the comparison is to evaluate if the extended hazard analysis technique, STPA-Teaming, identifies causal factors associated with collaborative control that were not previously considered.

Table 5-7 describes the scope of the two analyses. The baseline only includes the UCAs and scenarios relevant to the Team Lead (TL) – UAS collaboration analyzed in the case study (i.e., those in the blue region of Figure 5-1). The case study only counts the *unique scenarios*, which are not traced to or derived from other scenarios identified earlier in the analysis. The values in the table illustrate the number of data points available to determine whether STPA-Teaming identifies causal factors associated with collaboration that are not systematically found using STPA.

Table 5-7. Number of UCAs and Causal Scenarios in the Compared Analyses

Baseline STPA Analysis		Extended STPA-Teaming Analysis	
UCAs (in scope of comparison)	37	19 39	Abstracted UCCAs Refined UCCAs (High Priority)
N/A	-	67	Top-Level Scenarios
Higher-level scenarios (in scope)	89	313	Higher-level scenarios (unique)
Lower-level scenarios (in scope)	174	95	Lower-level scenarios (unique)
Total Scenarios (higher + lower)	263	408	Total Scenarios (higher + lower)

The numbers on both sides are similar, which suggests that the two analyses are reasonably balanced relative to one another in breadth and depth. In addition, the two studies are grounded in the same losses, hazards, controller responsibilities, and assumptions as specified in the baseline (see Section 5.1). These are both important considerations to make a fair comparison. The next subsections outline the differences and overlaps between the two techniques in identifying unsafe control and causal scenarios.

5.5.1 Comparing the Identification of Unsafe Control

To explain why some causal factors are found only in one of the techniques and not the other, it is important first to examine the differences in unsafe controls considered by both methods. The unsafe combinations of control actions (UCCAs) in the extension are defined at a higher level of abstraction than the simpler UCAs in STPA. As such, the two are not directly comparable.

Instead, the top-level scenarios in STPA-Teaming, which explore control actions internal to the team (see Chapter 4.3), provide a better way to highlight how the techniques differ in focus in analyzing unsafe control. Table 5-8 outlines the numbers related to unsafe control items uncovered in both studies.

Table 5-8. Summary of Unsafe Controls Identified in Both Analyses

Baseline STPA Analysis		Extended STPA-Teaming Analysis	
UCAs (in scope of comparison)	37	67	Top-Level Scenarios
Subset* not covered by extension	10	40	Subset* not covered by baseline
Uniqueness of scenarios N/A	-	26	Subset* that lead to unique scenarios

**Subset of the row directly above*

The table shows that 40 of the 67 top-level scenarios uncovered in the case study cannot be reasonably traced to a UCA or a causal scenario in the baseline. In refinement, 26 of those produce

unique scenarios. Similarly, 10 of the UCAs analyzed in the baseline are not covered by the top-level scenarios or the refined causal factors. The uniqueness of the scenarios in the baseline was not evaluated.

Table 5-9 exemplifies the differences and overlaps in a subset of unsafe controls identified by both techniques. Appendix 3 includes the full dataset and annotates whether or not each top-level scenario in the full analysis was also found in the baseline.

Table 5-9. Comparison of Select Unsafe Control Identified by Both Techniques

Found in	#	STPA-Teaming Top Level Scenario* or Baseline STPA UCA**
Extension Only	1	S-3.3.3*: a UAS does not provide a fix despite being tasked and another UAS fires as tasked when the target fired on must be fixed.
	2	S-15.1.6*: TL tasks a UAS to end providing a fix early and tasks another UAS to start firing late relative to each other when the target fired on must be fixed.
	3	S-42.6.5*: TL fires and tasks a UAS to provide the fire command when that creates excessive damage.
Extension & Baseline	4	S-38.2.2*: TL tasks a UAS to fix that is not capable when another controller is. UCA 2**: TL provides "fix on target" command to the wrong UAV.
	5	S-43.7.3*: TL tasks the UAS team to search, but no UAS searches when no targets have been found. UCA 29**: Autonomous Controller does not implement a task from the TL
	6	S-3.3.1*: TL tasks a UAS to fire and does not task any UAS to fix when the target fired-on must be fixed. S-3.3.5*: TL fires and does not task any UAS to fix when the target fired-on must be fixed. UCA 22**: TL provides fire command before the right target has been targeted.
Baseline Only	7	UCA 14**: TL provides the track target command when the targets are already out of range or in a restricted area.
	8	UCA 23**: TL provides the fire command early before receiving authorization from higher authorities.
	9	UCA 38**: Autonomous Controller releases a missile after target is out of range

The first three examples in Table 5-9 illustrate the types of unsafe control combinations not systematically considered using STPA. Example 1 explores why some of the controllers tasked to work together do not contribute to the joint effort. Example 2 analyzes how multiple controllers are tasked together in an unsafe way. Example 3 investigates how a controller tasks a collaborator in a way incompatible with its own execution. These top-level scenarios all produce refined scenarios not found in STPA.

The next examples highlight some of the overlaps in the two analyses. The baseline explores cases when the wrong controller on the team is tasked (Example 4). It also generically looks at why a controller does not provide a command as tasked (Example 5). In most of these cases, STPA still only considers one type of controller and one type of command at a time. The broader

scope of unsafe control specified in STPA-Teaming allows it to later uncover causal factors in these examples that are not found in the baseline, as shown in Section 5.5.2.

Conversely, Example 6 illustrates how STPA can consider multiple control actions if the analyst includes some of them in the context. This is the case in baseline UCA 22, in which a controller provides a control action (firing) that is unsafe in the context of another control action (not providing a fix). However, unlike in STPA-Teaming, this consideration is not systematic, and therefore similar combinations of different control actions are missed.

There are several other limitations with how the baseline handles UCA 22. Most importantly, the causal scenarios focus only on why the first control action (firing) would occur in that context. It does not analyze the second control action (not providing a fix) in the context of the first. As such, the interaction in Example 1 of Table 5-9 and others like it are missed. In addition, UCA 22 is ambiguous regarding which controller actually provides the fire command, as it could represent the TL firing directly or the TL tasking a UAS to fire. In STPA-Teaming, two different top-level scenarios are systematically created to cover these two cases, and each is then refined into its own *unique scenarios* not found in the baseline.

The last three examples in Table 5-9 demonstrate UCAs that are found in the baseline but not in the extension. The reason these are missed relates to the different focus of STPA-Teaming, which concentrates primarily on how multiple control actions are unsafe relative to one another. As such, it can miss contexts in which any single control action is unsafe, which is the focus of STPA.

The results in this section suggest that STPA and STPA-Teaming complement each other. A method to navigate between the two techniques is introduced in Chapter 6.

5.5.2 Comparing the Identification of Causal Scenarios

The loss scenarios and their causal factors are the ultimate outputs of STPA and STPA-Teaming. The information they provide helps specify safety constraints that inform system design. This section highlights the new causal factors that are uncovered in the MUM-T case study using STPA-Teaming, which are not found in the baseline.

Method of Comparison

Scenarios in STPA-Teaming are uncovered by systematically considering different types of causal factors in the refinement process (see Figure 4-21). These relate to internal feedback control factors (Step 2), collaborative control factors (Step 3), and other factors (Step 4). Step 3 of this process addresses five of the eight collaborative control dynamics exhibited by the MUM-T system. These are *cognitive alignment*, *lateral coordination*, *mutually closing control loops*, *dynamic membership*, and *dynamic connectivity*.

For these five dynamics, the evaluation compares each *unique scenario* to all the scenarios developed in the baseline analysis. Any *unique scenario* that contributes new information that cannot be reasonably related to any finding in the baseline is tracked as a *new scenario.* However, the goal is to offer the baseline the "benefit of the doubt" to ensure that the new causal factors are truly not found by STPA.

For this reason, several heuristics are employed to avoid over-counting *new scenarios* in STPA-Teaming. First, significant leeway is afforded to match any of the factors in the overall baseline to any one specific scenario found in the STPA-Teaming analysis. For example, if a factor in the baseline associated with a UAS providing the *fix* command is reasonably relevant to a scenario in the new analysis involving the TL providing the *fire* command, then no new scenario is designated.

In addition, any one baseline factor is allowed to count against multiple different *unique scenarios* in the new analysis. Finally, only *unique scenarios* from the STPA-Teaming analysis are included in the comparison. The derived scenarios, which still can provide new useful design information, are excluded.

As described in Chapter 4, the other three collaborative control dynamics are addressed as part of the UCCA identification process. Specifically, in this case study, *dynamic authority* applies to Type 1-2 UCCAs found using Abstraction 2b (combination of controllers issuing a common control action). *Transfer of authority* is addressed in Abstraction 2b Type 3-4 UCCAs, as those focus on handoffs. Finally, *shared authority* is exhibited in all UCCAs, but most uniquely applies to Abstraction 2a UCCAs (combinations of different control actions provided by the team).

For these three dynamics, the evaluation focuses on *unique scenarios* found in refinement Steps 2 and 4 since they exclude the other collaborative control dynamics addressed in Step 3. The same heuristics are used to not overcount *new scenarios*.

In addition, the comparison excludes refined scenarios developed in Step 2 if the top-level scenario is covered by a UCA in the baseline. In such cases, the evaluation conservatively assumes that those scenarios could have been found in the baseline, even if they were not, because the causal factors considered in Step 2 come from STPA.

Quantitative Comparison of Scenarios Found Using Each Technique

System-theoretic approaches encourage analysts to abstract concepts as needed to manage complexity. As such, in both analyses, causal scenarios are developed at different levels of abstraction. A higher-level scenario may contain broad information that can be refined into many lower-level scenarios, and those may, in turn, be further refined, and so on. The various levels are not necessarily aligned across scenarios within each analysis. This means that what is considered higher-level in one scenario may be similar to a lower-level scenario elsewhere.

The different levels of abstraction make it impractical to use the number of scenarios as a quantitative measure to compare the amount of information gained in the two analyses. As such, the comparison must be qualitative in focus. However, the number of scenarios uncovered illustrates the scope of the two analyses. They also help show that multiple causal factors were found in the extension that were not considered in STPA.

Table 5-10 outlines the number of *unique scenarios* in the extended hazard analysis that are both not found (new) and found in the baseline. It provides two important takeaways. First, multiple *new scenarios* are found using STPA-Teaming by considering each of the eight collaborative control dynamics exhibited by the system. These scenarios have causal factors that are not systematically found in the baseline using STPA.

Second, the baseline does find causal factors that align with scenarios found for each of these collaborative control dynamics. This is important because it again suggests a reasonable balance

between the two analyses. The remodeled case study and the extensions do not steer the analysis down a tangent completely unrelated to the baseline.

Table 5-10. Number of Unique Scenarios Not Found / Found in Baseline

Collaborative Control Dynamic	New: Not Found in Baseline	Previously Found in Baseline
Cognitive Alignment*	75	34
Lateral Coordination	29	4
Mutually Closing Control Loops	36	4
Dynamic Membership	25	5
Dynamic Connectivity	13	5
Transfer of Authority**	6	7
Dynamic Authority**	15	7
Shared Authority**	41	23
Total	240	89

** Excludes Lateral Coordination scenarios embedded in Cognitive Alignment*

*** Excludes scenarios refined in Step 3, which focus on the first five collaborative dynamics*

The numbers in Table 5-10 are the sum of the higher-level and lower-level scenarios in each category. If a higher-level scenario was found in the baseline, its lower-level scenarios were not considered in the count.

The remainder of this section provides examples of causal scenarios that relate to the eight collaborative control dynamics counted in Table 5-10. In most cases, the examples include both new scenarios not found in the baseline and scenarios from the baseline that contain similar causal factors. The qualitative discussion aims to contrast the types of scenarios created by the new technique with those developed using baseline STPA.

Qualitative Discussion: Cognitive Alignment and Lateral Coordination

Table 5-11 highlights example scenarios that relate to lateral coordination and cognitive alignment. Some of these scenarios are taken from Example 1 in Section 5.4 to show how they fit into the broader analysis.

There are several reasons why the baseline analysis does not find the scenarios listed. First, and most importantly, the baseline control structure does not represent the controlled process shared between the controllers on the team (i.e., Mission Processes). As such, it is difficult to systematically recognize that there is interdependence between the fix and fire control loops, regardless of which controller provides those control actions. This coupling, in this case, necessitates lateral coordination to occur. Both controllers must be able to bilaterally influence one another by sharing status information (e.g., "2 min to start", or "ready").

Second, information shared in lateral coordination can be obfuscated in a hierarchal control structure. Because the TL is hierarchically superior to the UAS, STPA typically treats all the information flowing down from the TL to the UAS as control actions. While the TL does control the UAS by providing tasking commands, at a lower level the TL and UAS must also coordinate as illustrated above. By recognizing lateral coordination can occur between the two levels, the

extended hazard analysis can reason about how inadequate information flow, beyond control, contributes to the unsafe collective control output.

Third, and related to cognitive alignment in general, the analysis does not consider just the process model of the controller that issues the fix command. It considers the process models of all the controllers on the team together. This is why in the last new scenario example, a controller that is not necessarily involved in the control loop but has an inadequate process model of the shared control process, has the ability to negatively influence the controllers directly involved.

Table 5-11. Examples Comparing Cognitive Alignment & Lateral Coordination Scenarios

UCCA 37.1: Controller Ci does not fix and no other Cj fixes when there is a priority target to engage and a teammate able to fire [H3] **Top-Level Scenario S-37.1.3**: TL adequately directs UAS, but some of the UAS do not execute directions properly. Here, the TL tasks a single capable UAS to fix, but the UAS does not fix.	
New: not in baseline	• Lateral coordination between UAS_1 (tasked to fix) and TL (to fire) is hindered by human-machine asymmetry in information semantics. Both controllers interpret shared information differently… (see full scenario in Example 1, Section 5.4) • Lateral coordination between UAS_1 (to fix) and TL (to fire) is hindered by human-machine asymmetry in information timing… (see full scenario in Example 1, Section 5.4) • UAS_2 is temporarily disconnected and its model of the target diverges from that of the team. UAS_2 reconnects and now contributes its divergent model variables. This disturbs team consensus regarding how to engage the target, changes the process model of UAS_1 assigned a task, and contributes it to not do it.
Found in baseline	• The information controllers use to coordinate the coupled fix-fire task is inconsistent. For example, UAS_1 (tasked to fix) receives state estimate from UAS_2 (to fire) that becomes slightly outdated due to small communications and processing delays. It compares the now slightly outdated UAS_2 state estimate with its own current UAS_1 state, and incorrectly determines that UAS_2 lags too much to provide the fix command. The same effect also impacts UAS_2 plans to fire.
Baseline Scenario	**Scenario**: The Autonomous Controller has incorrect feedback about the UAV's attitude, altitude, velocity, etc. [So it does not implement a task from the TL] • The FMS is sending data delayed so the Autonomous Controller is calculating states behind where the UAV is actually located or its current speed

The table also includes an example of a scenario that was found in the baseline. Fundamentally the scenario involves delayed state feedback that contributes to the unsafe control provided by a UAS. However, the scenario in the extended analysis goes further and explores how delays in receiving state information from collaborating controllers can affect joint execution. As such, the delays are again assessed as problematic across multiple controllers.

Qualitative Discussion: Mutually Closing Control Loops

Table 5-12 provides examples related to the *mutually closing control loops* dynamic. Some of the scenarios are from Example 1 in Section 5.4 and relate to the same UCCA in the cognitive alignment discussion above.

Table 5-12. Examples Comparing Mutually Closing Control Loop Scenarios

New: not in baseline	**UCCA 37.1** and **Top-Level Scenario S-37.1.3** same as in Table 5-11 • *Feedback about Controlled Process from Collaborators*: The UAS tasked to fix (e.g., UAS_1), prior to starting the fix, receives incorrect feedback from a teammate that the current fix is adequate to fire. The teammate is broadcasting feedback for a different fix, provided by a different UAS, for a different target. … (See full scenario in Example 1, Section 5.4) • *Feedback about Collaborator Control Actions from Controlled Process*: The UAS tasked to fix a target (e.g., UAS_1) temporarily receives fix energy off the target provided by another UAS (e.g., UAS_2). This leads UAS_1 to drop the task in the belief that UAS_2 is executing it… (See full scenario in Example 1, Section 5.4)
Found in baseline	**UCCA 40.4**: Controller Ci does not fire and no other Cj fires when there is a priority target to engage and a teammate able to fix [H3] **Top-Level Scenario S-40.4.5**: TL control actions to the process are unsafe with how it directs UAS. Here, TL does not fire and does not task a UAS to fire. • *Feedback about Collaborator Control Actions from Controlled Process*: TL does not receive the fix feedback necessary to close the fire loop that should be provided by a collaborating UAS. Similarly, the UAS does not receive feedback from the TL that the fix it is providing is inadequate.
Baseline Scenario	**Scenario**: Autonomous Controller receives feedback about the target, but cannot identify the target designation, so [it does not release missile as intended]

In scenario development, STPA tends to focus on the feedback control loop that involves the controller providing the UCA and the controlled process. In the *mutually closing control loops* dynamic, the feedback loops are closed through multiple collaborating controllers. Without this consideration, STPA does not systematically consider how the feedback necessary for a controller to provide its control actions is received from a collaborator. Similarly, it does not systematically explore how control actions provided by a collaborator impact the feedback received by a controller. This is why both of the top scenarios in Table 5-12 are missed in the baseline.

Conversely, the bottom scenario in Table 5-12 shows how STPA can find causal factors associated with this dynamic if the analyst has the creativity to reason about it. However, as previously stated, the lack of specific guidance regarding this type of interaction leaves the technique vulnerable to missing this factor. Furthermore, the scenario in the baseline is more focused on a single controller, whereas the STPA-Teaming scenario considers the multiple controllers involved.

Qualitative Discussion: Dynamic Membership

Table 5-13 has examples of dynamic membership causal scenarios. Most of the causal factors not found in the baseline are related to changes in the set of controllers involved in joint control, similar to the example listed. Conversely, out of the few related causal factors that were found in the baseline, most were focused on a variable set of subprocesses to control.

Table 5-13. Comparison Example of Dynamic Membership Scenarios

UCCA 1.1: The Team does not provide the fix, fire, or search commands when there are mission tasks to execute [H3] **Top-Level Scenario S-1.1.2**: TL directs UAS in a way that leads to unsafe collective control. Here, the TL selects incorrect controllers for each task.	
New: not in baseline	• *Dynamic Membership:* TL tasks two UAS on the team to fix and search. UAS_1 is assigned to fix and both UAS receive the search tasking, which is rated as a higher priority than the fix. UAS_2 is assigned by the automation to search but later drops out of the team (e.g., taken offline by Ground Station). The autonomy reassigns UAS_1 to the higher priority search task, which now opens the fix task. Then UAS_2 rejoins, and the TL proactively assigns it to provide the fix. The autonomy falsely assumes UAS_2 will resume the search, and releases UAS_1 back to its original task of providing a fix. Because two UAS are now assigned to fix, they both drop the task and no controller fixes nor searches.
Found in baseline	• *Dynamic Membership:* The set of mission tasks is highly dynamic. By the time the TL tasks a UAS to fix or fire, that task is no longer relevant. However, the autonomy applies the previous tasking to the next target that appears, against TL intent (related to Unsafe Control Path). In parallel, TL tasks another UAS to fix or fire on the new target, which creates a conflict.
Baseline Scenario	**Scenario**: TL initially authorizes a fix on target command. There is a delay in another command from the TL to delete the command. AuC receives a fix on target command without authorization.

In several cases, the factors found in the baseline were not directly related to dynamic membership. However, it is by considering this type of interaction that those causal factors were found in STPA-Teaming, as is the case in the example provided. This is why the baseline scenario is counted as part of this dynamic. Similar situations occur in the baseline scenarios related to the other collaborative control dynamics.

Qualitative Discussion: Dynamic Connectivity

STPA does consider dynamic connectivity, in part, by exploring unsafe control paths and feedback paths. Several such scenarios found in the baseline are listed in Table 5-14. STPA-Teaming goes further by exploring how the network dynamics can lead to unsafe control. The example in the table exemplifies this situation and is beyond what STPA typically examines for unsafe paths in a single feedback loop.

The top-level scenario listed in Table 5-14 is aligned to a UCA found in the baseline. This example demonstrates how, in such situations, the extended hazard analysis is still able to uncover causal factors that are not considered in the analysis of the UCA in the baseline. This is because the extension systematically investigates causal interactions that occur in collaboration, beyond those in simpler feedback control that are found in STPA.

Table 5-14. Comparison Example of Dynamic Connectivity Scenarios

New: not in baseline	**Top-Level Scenario S-38.2.3**: TL adequately directs UAS, but some of the UAS do not execute directions properly. Here, TL tasks a UAS to fix, but another incapable UAS fixes. • *Dynamic Connectivity:* UAS_2 is indirectly connected to TL via UAS_1 as a relay. When TL tasks UAS_2 to fix via UAS_1, UAS_1 misinterprets the message as it being assigned the task. As a result, UAS_1 provides the fix, and UAS_2 does not.
Found in baseline	**Top-Level Scenario S-37.1.1**: TL does not direct the UAS as necessary for the team to provide safe collective control. Here, TL does not task any UAS to fix. • *Unsafe Control Path:* TL intends to task a UAS but is unable to do so. The communication channel to do so is currently inadequate. This could be due to RF jamming, communications fading, misconfigured encryption settings, or other…
Baseline Scenario	**Scenario**: TL provides "fix on target" command, but the Autonomous Controller does not receive it. This can occur because: Autonomous Controller is out of range from the TL, there is not enough power to transfer the command, or the command is corrupted in transmission.

Qualitative Discussion: Dynamic Authority, Transfer of Authority, Shared Authority

Dynamic authority, *transfer of authority*, and *shared authority* are addressed earlier in STPA-Teaming as part of the identification of UCCAs. These interactions can influence any of the scenarios created later in the process, including all of the previous examples, which are found when the scenario refinement process focuses on collaborative control (Step 3 of Figure 4-21).

To focus more directly on causality related to shared, dynamic, and transfer of authority, the examples below only include scenarios developed in Steps 2 and 4 of the refinement process. As a reminder, Step 2 explores causal factors associated with internal feedback control loops. Step 4 analyzes other factors beyond the team interactions, which include unsafe feedback paths from the process, unsafe control paths to the process, and unsafe shared process behavior.

Table 5-15 highlights scenarios traced to *dynamic authority*. For the MUM-T case study, these scenarios are derived from Type 1-2 UCCAs found using Abstraction 2b (combination of controllers issuing a common control action).

Table 5-15. Comparison Example of Dynamic Authority Scenarios

UCCA 41.5: Controller Ci fires and no other Cj fires when Ci is not capable and a Cj is [H3]	
Top-Level Scenario S-41.5.5: TL control actions to the process are unsafe with how it directs the UAS. Here, TL fires her/himself when they are not capable to, instead of tasking a UAS that can.	
New: not in baseline	• *Inadequate Process Model: Inadequate feedback.* TL interface hides the TL's own aircraft status and capabilities behind that of the UAS (or vice versa) and makes it difficult to consider all resource options available on the team.
Found in baseline	• *Unsafe Control Path:* TL weapon system malfunctions and fires unintentionally. The UAS that was assigned to the task observes this and drops its task.
Baseline Scenario	**Scenario**: The UAS [or any controller] does not intentionally release a missile, but there is electrical interference that causes the bay to unlock, or a missile is loose in the bay, so a missile is released.

The example of the scenario not found in the baseline illustrates a key strength of STPA-Teaming. Instead of looking separately at why the TL does not fire, or why a UAS does not fire, the analysis explores why none of the controllers together are able to fire. This highlights a need for the TL's mental model to consider her/his own capabilities and that of the UAS together. STPA will point to mental model gaps regarding the TL or the UAS capabilities, but it may not emphasize the mental model gap of the collective team capability.

The same idea also applies to the causal factor that was found in the baseline, which fundamentally relates to a weapon malfunction in both analyses. However, the STPA scenario is again more focused on one controller, whereas the extension considers how the malfunction affects the collaborators as well.

Examples related to *transfer of authority* are shown in Table 5-16. In this case study, these scenarios are associated with Abstraction 2b Type 3-4 UCCAs, in which controllers start and end a common control action in an unsafe sequence. Because STPA does not systematically consider this type of control relationship, none of the top-level scenarios produced in STPA-Teaming are addressed by UCAs in the baseline.

But as was explained previously, the baseline does find lower-level causal factors that relate to some of the transfer of authority scenarios found in the extension. In the example provided, both analyses recognize that the TL has an inadequate mental model of the UAS execution timeline. However, the scenario from the extension considers the implication of this gap for multiple controllers working together.

Table 5-16. Comparison Example of Transfer of Authority Scenarios

UCCA 47.1: Controller Ci starts providing fix command before Cj ends providing fix command when that creates mutual interference [H1, H3] **Top-Level Scenario S-47.1.6**: TL directs UAS in a way that leads to unsafe temporal sequencing. Here, TL tasks a UAS to start fix too soon and another to end fix too late relative to each other.	
New: not in baseline	• *Inadequate Control Algorithm:* TL has accurate information about the target and the execution timelines of controllers on the team, but s/he still chooses to task the UAS to issue the fix with excessive overlap. *Refinement*: o TL has a misunderstanding of how the UAS operate, and s/he believes that the UAS will coordinate with one another to avoid the unsafe handoff.
Found in baseline	• *Inadequate Process Model:* TL incorrectly believes (1) the time windows tasked do not excessively overlap, (2) the UAS will not provide a fix over their entire time window, (3) multiple UAS will not cause hazardous effects if they fix simultaneously. *Refinement*: o *Missing Feedback.* TL interface does not provide a way to effectively compare UAS tasking timelines.
Baseline Scenario	**Scenario**: TL has accurate feedback related to timing but has an inaccurate mental model of how long it takes the Automated Controller to implement a command upon receiving it, so TL issues the command late.

The last of the dynamics discussed in the case study is *shared authority*. This dynamic is fundamental to all teaming interactions modeled in the generic collaborative control structure (Chapter 4.1). As such, it influences all UCCAs and all scenarios. For the MUM-T case study, the scenarios developed from Abstraction 2a UCCAs (combinations of different control actions provided by the team) and refined in Steps 2 and 4 are the most uniquely related to shared authority. Table 5-17 provides some examples.

In this case, two scenarios found in STPA-Teaming are refined from the same higher-level scenario. The example found in the baseline illustrates how STPA can create scenarios that involve multiple control actions if the analyst is able to reason about this complexity. However, the lack of a systematic approach on this point is why the baseline only finds one of the two scenarios identified by the extension.

Table 5-17. Comparison Example of Shared Authority Scenarios

UCCA 3.3: Team fires but does not provide a fix when target fired-on must be fixed. [H3, H5] **Top-Level Scenario S-3.3.1**: TL does not direct the UAS as necessary for the team to provide safe collective control. Here, TL does not task a UAS to fix but s/he fires or tasks a UAS to fire.	
New: not in baseline	• *Inadequate Control Algorithm:* TL has accurate information about a controller intending to fire and no controller is tasked to provide the coupled fix, but s/he still chooses not to task the coupled command. *Refinement*: o TL misunderstands how TL or UAS weapons systems operate. S/he incorrectly believes that if there is no fix, the weapon system will not fire.
Found in baseline	• *Inadequate Control Algorithm:* same as above. *Refinement*: o TL misunderstands how the UAS operate. S/he incorrectly believes that if a UAS is approved to fire, the UAS automatically provides the coupled fix.
Baseline Scenario	**Scenario**: TL does not provide fire [or fix] because he believes the UAV(s) can automatically fire upon identifying targets. TL assumes system works similarly to other MUM-T systems where UAV(s) have automatic firing [or fix] capabilities.

5.6 Dynamic Hierarchy Demonstration

As shown in Figure 5-2, the MUM-T system modeled in the baseline exhibits all of the collaborative control dynamics defined in Chapter 3 except for *dynamic hierarchy*. Up to this point, the case study has remained consistent with the baseline to compare the results of STPA-Teaming with those produced using STPA. In this section, the baseline assumptions are relaxed to incorporate *dynamic hierarchy* and demonstrate how the new technique handles it.

The system is modified as follows. In the baseline concept, the human Team Lead (TL) tasks the UAS team as a whole to execute the search task. In effect, the TL delegates to the autonomy the authority to allocate that task to one of the UAS. In a modified concept of MUM-T, the autonomy could include the TL as one of the controllers, in addition to the UAS, to which it can assign the task. This creates dynamic hierarchy, as the TL generally oversees the UAS team, but at a lower level the UAS can command the TL to go search.

The remainder of the section demonstrates of how the extended hazard analysis identifies causal factors related to dynamic hierarchy. It is not a full analysis. Causal scenarios are developed for only two of the relevant Unsafe Combinations of Control Actions (UCCAs). However, the effort reflects the type of safety analysis that can be performed when exploring different conceptual design alternatives.

The first step of the hazard analysis is unchanged from the original case study. The losses, hazards, and system boundary remain unchanged from those introduced in Section 5.1. In the next step, the collaborative control structure of the system is updated to reflect the ability of the UAS to retask the search to the TL, as shown in red in Figure 5-8.

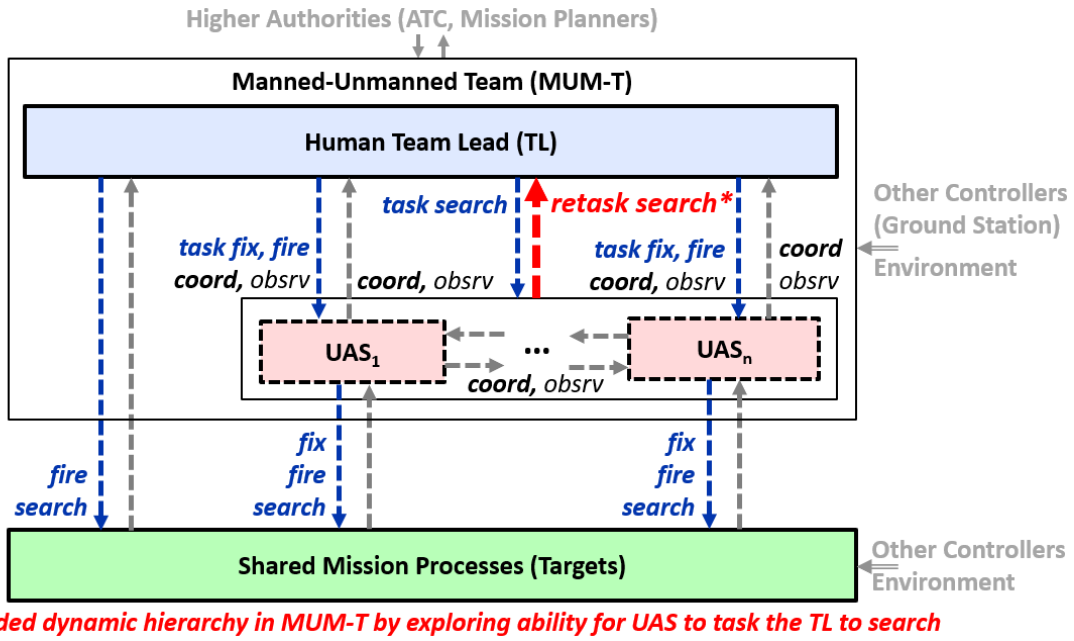


Figure 5-8. Modified MUM-T Control Structure that Includes Dynamic Hierarchy

No change occurs in the third step of the analysis. The UCCAs, which represent the collective control output of the team, remain the same as those generated in Section 5.3. The modification to the system only impacts the UCCAs that involve the search command, which are consolidated into Table 5-18. The refinement for each of those UCCAs is also unchanged.

Table 5-18. UCCAs Relevant to Dynamic Hierarchy Analysis Demonstration

id	controllers		context
	team	team	Abstraction 2a UCCAs
1.1	¬fix	¬{fire,search}	*when there are mission tasks to execute [H3]*
10.5	search	¬{fix,fire}	*when engaging known target is higher priority than searching [H3]*
11.6	¬search	{fix,fire}	*when searching for another target has higher priority [H3]*
	Ci	Cj(s)	Abstraction 2b UCCAs
43.7	¬search	¬search	*when no targets have been found [H3]*
44.8	search	¬search	*when entity needed for priority task and teammate can search [H3]*
56.3	E(search)	**F** S(search)	*when that creates an excessive gap in the search [H3]*

In the fourth step of the analysis, the causal scenarios are developed using the same process defined in Chapter 4.3. They are initialized with *top-level scenarios,* which describe the different possible control actions provided internally to the team that are relevant to the unsafe collective output. Dynamic hierarchy reflects internal control, so this is where it is addressed in STPA-Teaming. The addition of a new internal control action from the UAS to the TL generates new top-level scenarios. The scenarios are then refined using the same process defined in Figure 4-21.

Two examples follow to show the development of scenarios related to dynamic hierarchy. The first identifies all top-level scenarios for a Type 1-2 UCCA and refines one of them using the Figure 4-21 template. The second defines the top-level scenarios for a Type 3-4 UCCA. Appendix

4 contains the full set of dynamic hierarchy causal scenarios for these two UCCAs. The other UCCAs listed in Table 5-18 are not analyzed as part of the scope of this work, but their scenarios could be developed using the same method. The causal scenarios related to dynamic hierarchy are indexed as S_{DH} to differentiate them from the main case study in the previous sections.

Example 1. Dynamic Hierarchy Scenario Development for Type 1-2 UCCA

UCCA 43.7 (abstracted UCCA): Controller C_i does not search and no other C_j searches when no targets have been found [H3]

UCCA 43.7.1 (refined UCCA): TL, UAS_1, and UAS_n do not provide the search command

Figure 5-9 shows how the Team Lead (TL) and the UAS can task each other to search given the dynamic hierarchy. Each case is traced to its relevant top-level scenario(s). The third case is not logical in the context of the UCCA.

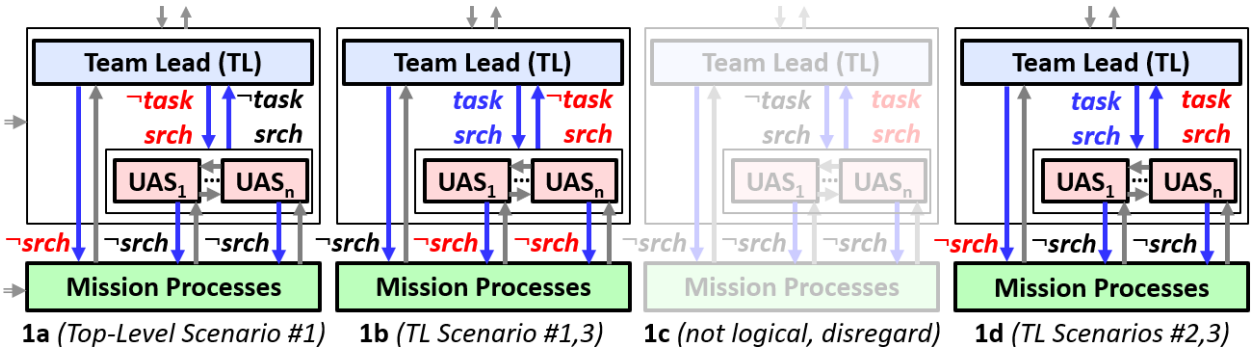


Figure 5-9. Internal Control Combinations that Result in No Controller Providing the Search

S_{DH}-43.7.1 (Step 1: Top-Level Scenarios #1): A controller does direct other controllers on the team as necessary for the team to execute safe collective control of the shared process. In the UCCA context:
(1) TL does not task UAS team to search (and does not search her/himself) (see old S-43.7.1)
(2) UAS team does not retask TL to search (and does not allocate a UAS to search) **[NEW]:**

- ***(Step 2: Internal Control) Unsafe Control Input:*** UAS team interprets direction from another controller (e.g., Ground Station, another TL, a cyber attacker) that it should never retask the search command to the TL, even if the TL is the only controller capable of searching. **[NEW]**
- ***(Step 2: Internal Control) Inadequate Process Model:*** UAS Team has inadequate process model variables: (1) does not believe it has authority to task TL, (2) believes that TL is intending to search, or (3) does not believe TL can search. **[NEW]** *Refined in Appendix 4…*
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** UAS are programmed to assume that the TL will search if a UAS is not allocated (instead of actively retasking TL to search). **[NEW]** *Refined in Appendix 4…*
- ***(Step 2: Internal Control) Unsafe Control Path:*** UAS Team intends to retask TL but is unable to do so. TL does not want to be interrupted by the UAS and blocks the control

path for that command. The UAS Team continues to send the retask command without knowing the TL is purposefully ignoring them. **[NEW]** *More listed in Appendix 4…*

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*:** *Not applicable.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*:**
    - *Initialization:* UAS receive different information regarding the availability of the TL. This could occur as TL periodically updates her/his plans given slight changes in their understanding of the mission. The UAS are updated across different exchanges. As a result, they do not reach consensus that the TL can be retasked to search. **[NEW]**
    - *More listed in Appendix 4…*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** The team requires some majority or unanimity among the automated controllers to retask the search to the TL. However, the set of controllers fluctuates too much to achieve this threshold, so the team is unable to retask. This could happen, for example, if a UAS is cycled repeatedly in and out of the team due to some anomaly. **[NEW]**
- **(Step 3: Collaborative Dynamic)** ***Dynamic Connectivity:*** UAS Team has consensus and its members broadcasts to the TL the retasking for the search. Then, another UAS, which was disconnected from the team, connects and shares its default values with no plans to task anything to the TL. As a result, the assignment is dropped. **[NEW]**

S_{DH}-43.7.2 (Step 1: Top-Level Scenarios #2): A controller directs other controllers on the team in a way that leads to unsafe collective control. In the UCCA context: the UAS team incorrectly retasks the TL, who cannot search, instead of allocating the task to a UAS. **[NEW]**

- *Refined in Appendix 4…*

S_{DH}-43.7.3 (Step 1: Top-Level Scenarios #3): A controller directs other controllers on the team adequately, but some of those controllers do not execute directions properly, which leads to unsafe collective control. In the UCCA context:
(1) TL tasks the UAS team to search, and no UAS searches (see old S-43.7.3)
(2) UAS team retasks the search, but the TL does not execute it **[NEW]**

S_{DH}-43.7.4 (Step 1: Top-Level Scenarios #4): A controller adequately does not direct others on the team to provide certain commands, but some of those controllers provide them anyways, which leads to unsafe collective control. *Not applicable in the UCCA context* (same as old S-43.7.4)

S_{DH}-43.7.5 (Step 1: Top-Level Scenarios #5): A controller provides control actions to the shared process that are unsafe in combination with how it directs other controllers on the team. In the UCCA context:

(1) TL does not search when s/he does not task the UAS team to search (see old S-43.7.5)
(2) UAS team does not allocate a UAS when it does not retask the TL to search (see old S-43.7.3)

S_{DH}-43.7.9 (Step 4: Other Factors): *Not specific to dynamic hierarchy* (see old S-43.7.9)

Example 2. Dynamic Hierarchy Scenarios for Type 3-4 UCCA (Top-Level Scenarios Only)

UCCA 56.3 (abstracted UCCA): Controller C_j ends search before C_i starts search when that creates a large gap in a search handoff [H3]

UCCA 56.3.1 (refined UCCA): TL ends search before UAS_1 starts search
UCCA 56.3.2 (refined UCCA): UAS_1 ends search before TL starts search
UCCA 56.3.3 (refined UCCA): UAS_2 ends search before UAS_1 starts search

S_{DH}-56.3.6 (Step 1: Top-Level Scenarios #6): A controller directs other controllers on the team in a way that leads to unsafe sequencing. In the UCCA context:
(1) TL tasks the UAS to start or end the search in a way that creates a gap with the TL's planned start or end to the search (see old S-56.3.6)
(2) UAS team retasks TL to start/end search in a way that creates a gap with the UAS's planned start or end to the search **[NEW]**

S_{DH}-56.3.7 (Step 1: Top-Level Scenarios #7): A controller adequately directs other controllers on the team, but those controllers execute the directions in a way that leads to unsafe temporal sequencing. In the UCCA context:
(1) UAS1 stops providing the search before UAS2 starts providing the search (see old S-56.3.7)
(2) The retasked TL starts/ends providing the searching too late/early in a way that leads to a gap in a TL-UAS handoff **[NEW]**

S_{DH}-56.3.8 (Step 1: Top-Level Scenarios #8): A controller provides control actions to the shared process that are unsafe in temporal sequencing with how it directs other controllers on the team. In the UCCA context:
(1) TL executes her/his search in a time window inconsistent with the one s/he tasked to the UAS for a TL-UAS handoff (see old S-56.3.8)
(2) UAS allocate a portion of the search to a UAS in a way that is inconsistent with how they retasked the other portion to the TL for a TL-UAS handoff **[NEW]**.

5.7 Summary

This chapter applied the extended hazard analysis technique developed in Chapter 4 to a manned-unmanned aircraft teaming aerospace system concept. The case study had two purposes. First, it demonstrated how to analyze the safety of a real-world system in development that exhibits the complex collaborative interactions defined in Chapter 3. Second, it helped evaluate whether STPA-Teaming is able to uncover new causal factors not identified by STPA.

The case-study started with the same losses and hazards defined in an independent STPA analysis of MUM-T that served as a baseline for comparison. Next, the system was remodeled using the generic collaborative control structure template introduced in Chapter 4.1. The new MUM-T model adheres to all the assumptions specified in the baseline, but it also highlights the eight collaborative control dynamics exhibited by the system.

The case study then identified Unsafe Collaborative Control Action (UCCA) using the algorithm developed in Chapter 4.2. This began with an automated enumeration of potential UCCAs at an abstracted level. The analyst then specified the context for the UCCAs that were deemed unsafe. Next, the automation refined, pruned, and prioritized those UCCAs. Finally, scenarios were developed using the process defined in Chapter 4.3 to identify causal factors related to collaborative control in the MUM-T system.

The results of STPA-Teaming were then compared to those provided by STPA in the baseline analysis. The new technique was able to find new unsafe control relationships and new causal scenarios that were not previously considered. The results were followed by a qualitative discussion of the differences in the findings between the two techniques.

Finally, the MUM-T case study was expanded beyond the baseline to incorporate dynamic hierarchy interactions into the system architecture. By applying STPA-Teaming on this design variation, the case study demonstrated how the new technique is able to handle all of the collaborative interactions defined in Chapter 3.

The demonstrated ability of STPA-Teaming to (1) find causal scenarios not found using STPA and (2) address all of the collaborative control dynamics defined in this work provide an argument to validate Hypothesis 3 of this dissertation.

> *Hypothesis 3: The STAMP and STPA extensions identify causal factors associated with collaborative control interactions, which are not systematically found using the existing STPA guidance.*

The next chapter introduces a framework to integrate STPA-Teaming into a systems engineering approach that supports safety-guided design. It builds on the MUM-T case study by deriving generalized safety constraints from the analysis in this chapter and by showing how those constraints can help integrate design decisions and assurance activities.

Chapter 6: A Framework for Safety-Guided Design of Collaborative Systems

Safety assurance is typically carried out separately from system design and in later stages of development. As discussed in Chapter 2, this practice is problematic. By the time assurance processes are applied, it is often too late to effectively modify a system to address safety issues that are found. In addition, the design may be so complex that it challenges the Verification and Validation (V&V) process to ensure hazards have been properly eliminated or mitigated. These shortfalls are exacerbated by the types of complex interactions sought in novel aerospace systems, and they are a major reason why these systems have not been fielded to date.

To overcome these problems, Leveson advocates for integrating system design and safety assurance activities from the earliest phases of development [4]. By conducting hazard analysis in early conceptual design, system requirements can be generated to address hazards by designing safety into the system from the beginning. This process can continue throughout the development lifecycle to assure the system is constructed to be safe. By doing this, final assurance activities like certification can consist of a review that hazards have been properly identified and handled throughout the design [4].

Chapters 3-5 develop and demonstrate the necessary foundation and techniques to analyze hazards rigorously and systematically for systems with complex team-inspired component interactions. What is now needed is a mechanism to incorporate the new analytical method, STPA-Teaming, into the design process. To this end, this chapter introduces a systems engineering framework that integrates safety-guided design with assurance activities through enhanced traceability.

The structure introduced is derived from Intent Specification [52], and it enables the traceability of information across three engineering axes relevant to system development. First, it aligns each model of a system with the hazard analysis, the derived safety constraints, and the selected V&V strategy. Second, it supports modeling the system at different levels of abstraction. In the case of collaborative systems, as emphasized in this work, the engineering focus shifts from the team as a whole, to interactions within the team, and then to lower-level component interactions. Third, the framework engineers the system using a means-end hierarchy, starting from concept of operation, down to low-level component implementation.

The remainder of this chapter is organized as follows. Section 6.1 details each of the three engineering axes of the framework. Section 6.2 demonstrates the top two levels of the framework on the Manned-Unmanned Teaming (MUM-T) system from the Chapter 5 case study. It explains how to conduct hazard analysis using STPA and STPA-Teaming on different model abstractions. Finally, Section 6.3 describes how the safety constraints derived from the hazard analysis can, through rigorous traceability, guide conceptual architectural decisions. While the discussion is focused on collaborative control, the general construct may be useful to more general systems.

6.1 Framework Description

The framework for safety-guided design of collaborative systems is shown in Figure 6-1. It is derived from the Intent Specification methodology, which was designed to enhance human problem-solving in the design of systems [52].

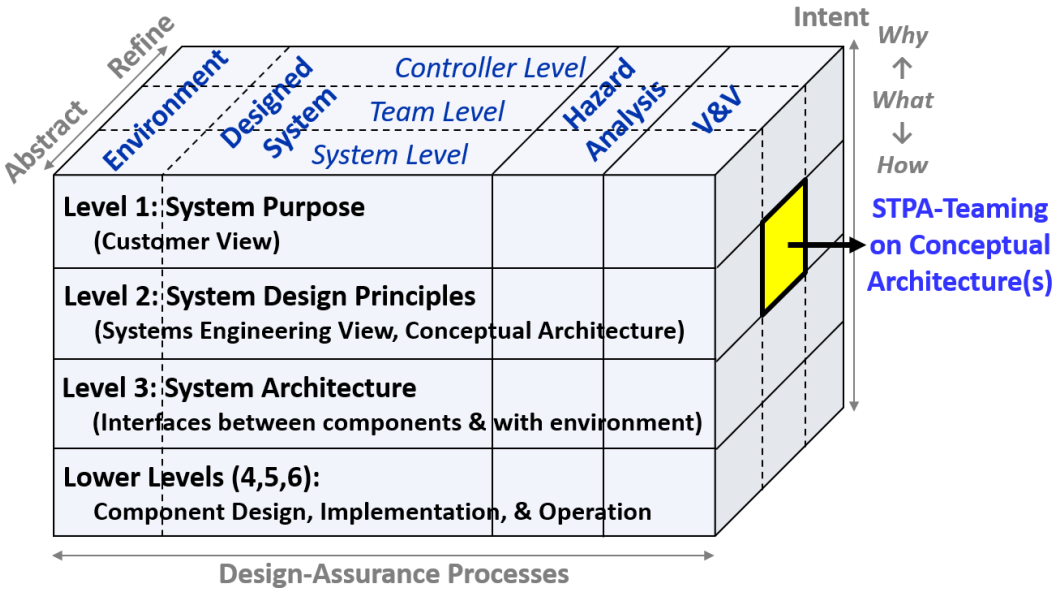


Figure 6-1. Framework for Safety-Guided Design of Collaborative Systems

The Intent Specification supports the systems engineering process of defining objectives, and iteratively developing, evaluating, and selecting design alternatives. Its content is governed by the principles of Systems Theory described in Chapter 2.4. Its structure organizes information in a way that improves information pickup in the face of complex system designs [52].

The general Intent Specification construct is tailored to formulate the framework introduced in this Chapter. Some of the implementation details on each axis are updated to reflect the author's vision for safety-guided design. A description of each axis follows.

6.1.1 Design-Assurance Processes Axis

The horizontal (width) axis of the framework traces the different processes involved in system design and assurance to one another. These processes can occur at different levels of abstraction (depth axis) and may vary for different engineering intents (vertical axis). However, they generally involve modeling, specification, analysis, and V&V activities.

The first two horizontal components in the framework, the *environment* and the *designed system*, are composed of system models and specifications. The system under design is modeled in its socio-technical context interacting with the environment. The designed system includes the components within the boundary over which there is engineering control, which includes human operators. The environment consists of all the components beyond this boundary, which constrain or influence the designed system.

The specifications define how a system must be built to achieve its goals within its constraints. Specifications include design requirements aimed to achieve various measures of performance. They also include design constraints, such as safety constraints, to prevent the system from exhibiting an unacceptable behavior. The rationale for the specifications must be traced to the assumptions or analyses that support them.

The *design-assurance axis* ensures that the safety constraints specified for the system are traced to the results of the hazard analysis from which they are derived. This work employs STPA and STPA-Teaming as the techniques to conduct hazard analysis for all of the reasons cited in Chapter 2. However, the framework is not restricted to those techniques, and can still be employed using other methods.

Finally, the specifications, including the safety constraints, are traced to the V&V strategy selected to ensure the system is built to specification (Verification) and that the specification is correct (Validation). More specific examples of some of the content on this axis are provided in Section 6.2.

6.1.2 Abstraction-Refinement Axis

The depth axis employs the System Theoretic foundation of the framework to model and analyze the system at different levels of abstraction. STAMP-based techniques emphasize starting at a high level of abstraction to analyze the system as a whole. Then, the model can be iteratively refined to add the detail necessary for the specification.

Figure 6-1 frames the levels of abstraction in the context of collaborative control systems. At the system level, a team of controllers is abstracted as a whole to explore how it collectively interacts with other controllers and the environment. At the team level the analysis focuses on interactions between the collaborators to examine how they contribute to the collective output of the team. Finally, the controller level explores interactions between any controller or its subcomponents with the rest of the system.

The *abstraction-refinement axis* is both iterative and recursive. Different views of the system may be represented at a similar level of abstraction to shift the analytical focus to different parts of the system, or to explore different design alternatives. Recursion is implied by Systems Theory, as teams are composed of controllers, teams may be composed of other teams, and controllers may be composed of teams. Section 6.2 demonstrates different model representations of the MUM-T system from the case study.

6.1.3 Engineering Intent Axis

The vertical axis traces the different types of specification information used in different engineering lifecycle activities. It employs a means-end hierarchy that describes for any level *what* the specification is, at the level above *why* the system is specified that way, and at the level below *how* the specification is implemented.

The levels are, to some extent, representative of activities that occur on the left side of the Systems Engineering "V" (Figure 2-2). However, this framework integrates at each level the assurance processes such as hazard analysis and V&V, which are more commonly considered only later in development, on the right side of the "V".

The first level represents the solution-neutral system to address the customer need. As described in Intent Specification, this begins by specifying the system goal, environmental assumptions, and environmental constraints that influence the system [52].

Next, high-level system requirements are derived from the system goals and may be supported by various systems engineering analyses, including business analysis, concept of operations analysis, and stakeholders analysis [215]. A V&V strategy must be considered for each requirement. At this level, verification may be conducted by construction of the verification activities performed at lower levels. Requirements may be validated by approval of a validation authority established by the customer [216].

The first level also includes safety constraints that are derived from the hazard analysis. At this level, the system is solution neutral so the hazard analysis is limited to the activities performed in Step 1 of STPA. These consist of identifying the losses and system-level hazards. As such, the safety constraints only include the system-level constraints derived from the hazards, also identified in STPA Step 1. The remainder of the hazard analysis is conducted at the levels below. As with the other requirements, safety constraints are also traced to a V&V strategy.

The second engineering intent level centers on the model, analysis, and specification of the conceptual architecture(s) for the system. As described in Chapter 2.2, a conceptual architecture represents the top-down control interactions of a system. Its analysis can help identify requirements early to enforce desired emergent properties before conventional architecting techniques are applied at the level below [101]. In this work, the conceptual architectures are instantiated using STAMP hierarchal control structures.

As described in the depth axis, multiple views of the control structure may exist to analyze the same system at different levels of abstraction. In addition, variations of the control structure may be included to explore different design alternatives to implement the costumer view specified at the level above. For example, the case study in Chapter 5 described two variants of the MUM-T conceptual architecture: one aligned with assumptions from previous work (i.e., the baseline [53]), and another added new control channels for the UAS to task the human pilot.

Each version of the control structure is traced to its hazard analysis, the derived safety constraints, and their selected V&V method. At this level, the hazard analysis is carried out using STPA, or STPA-Teaming, Steps 2-4. The technique selected depends on the level of abstraction being analyzed. At the system level, STPA analyzes the interactions of the collective team with the rest of the system and the environment. At the team level, STPA-Teaming examines the causal relationships between the collaborators on the team. STPA is again applied at the controller level to analyze sub-component interactions. This analytical sequence is exemplified in Section 6.2.

The third level of engineering intent describes how to implement the selected conceptual architecture using more traditional architecting techniques that "map forms and functions" [95]. A variety the methods that support this phase of engineering are described by Crawley et al. [98]. While the specific activities at this level are beyond the scope of this research, the framework still emphasizes the need to conduct hazard analysis on the more detailed architecture and to trace the specifications to V&V strategies.

Similarly, the lower levels of the framework, which are dedicated to component design, implementation, and operations to realize the architecture, are beyond the scope of this work. An example of the type of lower-level component detail they include is available in an example Intent

Specification for TCAS [217]. However, at these levels the framework continues to promote traceability between design and assurance activities and across different levels of abstraction.

6.2 Example Application of the Framework

The following is a short demonstration of the safety-guided design framework applied to the case study in Chapter 5. The MUM-T system analyzed is in conceptual design stages, so the following example focuses on the first two levels of engineering intent. The development of a full specification exceeds the scope of this dissertation.

6.2.1 Example: MUM-T Level 1 - System Purpose View

The following outlines the *Level 1 - System Purpose View* content for the MUM-T system. For brevity, it only includes a subset of each of the design-assurance elements. Some of the items are refined to illustrate how they are handled in the depth axis.

Environment

Environmental Assumptions:

- EA1: The MUM-T system will operate in environments where the performance of communication channels is degraded.
  - EA1.1: The MUM-T system will operate in the presence of hostile electronic warfare that interferes with the ability to communicate.
  - EA1.2: *list more items (beyond research scope) …*
- EA2: *list more items (beyond research scope) …*

Environmental Constraints:

- EC1: The MUM-T system must not interfere with civilian air traffic and airport operations.
- EC2: *list more items (beyond research scope) …*

Designed System

Designed System Boundary: Team Lead and aircraft, Ground Stations, UAS

System Goals:

- G1: Provide a military air strike capability equivalent to that provided by a flight of four human piloted modern fighter/attack aircraft, but with only one human pilot.
- G2: *list more items (beyond research scope) …*

System-Level Requirements (non-safety):

- R1.1: The MUM-T system must enable collaboration between one human piloted aircraft and four or more unmanned aircraft. [trace to supporting analysis]
- R1.2: *list more items (beyond research scope) …*

System-Level Safety Constraints: derived from hazards identified in STPA Step 1

- SC-1.1.1: Aircraft must satisfy minimum separation requirements from other aircraft and objects [H1] [53]
- SC-1.1.2: If aircraft violates minimum separation, then the violation must be detected, and measures must be taken to prevent a collision. [H1] [53]
- SC-1.2.1: Aircraft control must not be lost given any possible input provided by the flight controller. [H2]
- SC-1.2.2: If aircraft control is lost, the loss of control must be detected, and measures must be taken to regain aircraft control [H2]
- SC-1.3.1: The system must accomplish planned mission operations [H3]
- SC-1.3.2: If the system does not accomplish planned mission operations, the lack of accomplishment must be detected, and measures must be taken to regain the ability to accomplish those operations [H3]
- SC-1.3.3: If the system performed unplanned operations that violate rules of engagement, those operations must be detected, and measures must be taken to stop them. [H3]
- SC-1.4.1: Aircraft should not depart approved airspace [H4] [53]
- SC-1.4.2: If aircraft violates approved airspace constraint, then the violation must be detected and measures taken to prevent encounters with enemy [H4] [53]
- SC-1.5.1: The system must not fire at friendly assets or forces [H5] [53]
- SC-1.5.2: If the system fires at a friendly, then the violation must be detected, and measures taken to prevent impact [H5]

Hazard Analysis

At this level this only includes information from STPA Step 1

Losses: L1-L4 as listed in Chapter 5.1 and defined in baseline [53].

Hazards: H1-H5 as listed in Chapter 5.1 and defined in baseline [53].

Verification and Validation

Verification: by construction of levels below [SC-1, R1]

Validation: by approval of validation authority [SC-1, R1]

6.2.2 Example: MUM-T Level 2 - Conceptual Architecture View

Level 2 models and analyzes various conceptual architecture abstractions and design alternatives. Figure 6-2 exemplifies this process by presenting multiple abstractions of the MUM-T concept described in the baseline. This system actually includes two different teams of components. The first involves the Team Lead (TL) and the UAS collaborating to control the mission targets (Figure 6-2 left). This is the collaborative system analyzed in Chapter 5. The second team consists of the TL and the Ground Station (GS) that share control of the UAS (Figure 6-2 right).

For each of the two teams, the hazard analysis starts at the *system level* and uses STPA to explore how the team as a whole interacts with the rest of the system (Figure 6-2 top row). Next,

at the *team level*, the focus shifts to collaborative interactions between the controllers on each team. STPA-Teaming analyzes how those relationships contribute to the collective control output (middle-row). Finally, controllers are refined at the *controller level* to analyze, using STPA, how their component interactions can lead unsafe behaviors (bottom-row).

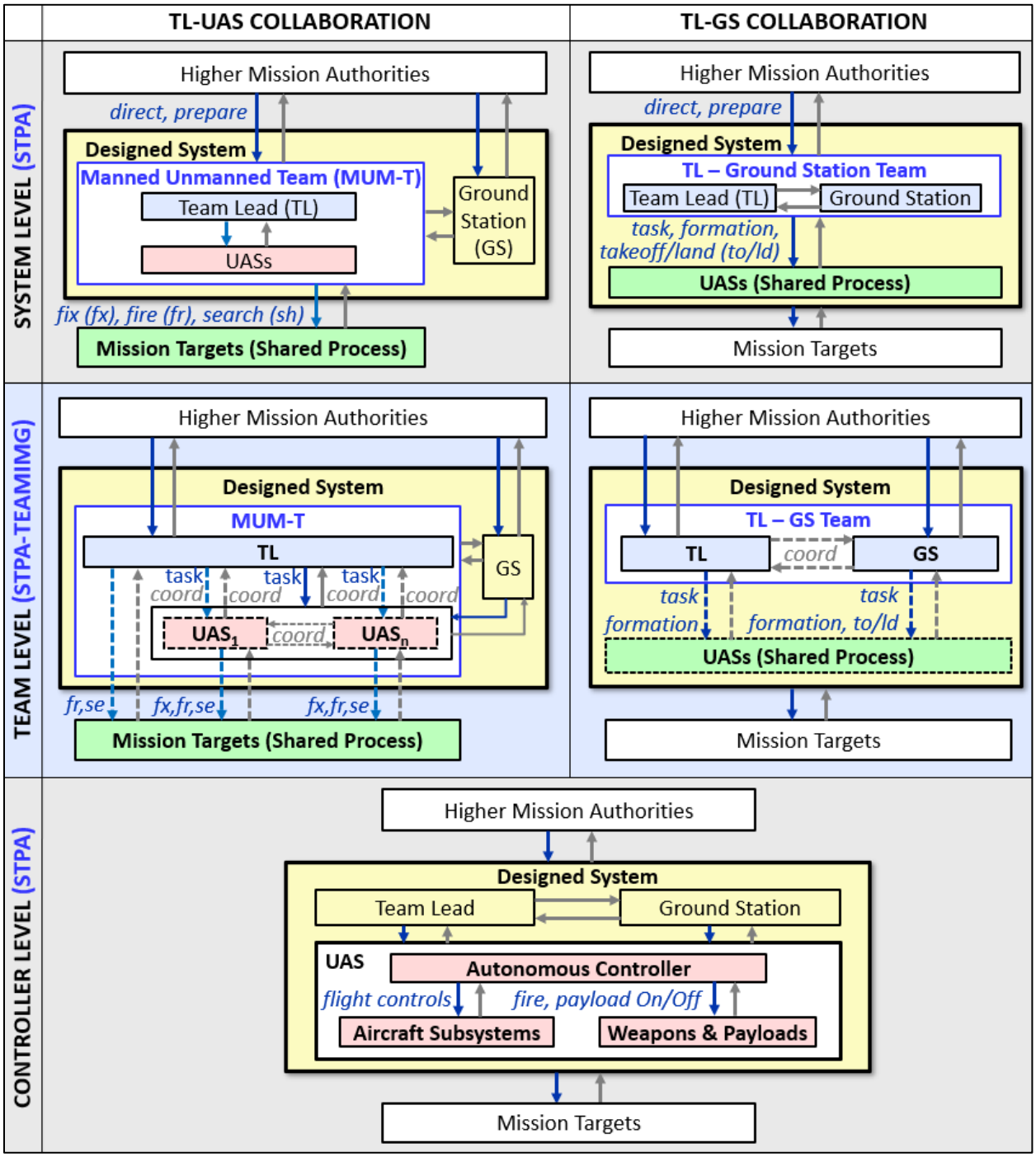


Figure 6-2. Different Abstracted Views of MUM-T Conceptual Architecture

Figure 6-2 designates which hazard analysis technique is applied to each control structure. In each case, the analysis focuses primarily on the control actions that are labeled, which are the

most relevant to the interactions emphasized by the abstraction. The goal is to produce a more complete analysis by selectively applying each technique to the types of interactions it is best equipped to handle.

Table 6-1 lists examples of the unsafe control actions (UCAs) and the unsafe combinations of control actions (UCCAs) found using each technique at each level. Each row includes an unsafe control item that was not found by the other analysis technique in the case study, as previously discussed in Table 5-9. In other words, the UCAs listed are part of the 10 items found to not be covered by STPA-Teaming. At the system level, the controller in these UCAs is reframed, from TL or UAS to MUM-T, to align with the level of abstraction at which they are explored. Similarly, the UCCAs are one of the many control relationships not considered in the baseline. The techniques are thus able to complement each other.

Table 6-1: Examples of UCAs and UCCAs Produced at Each Level of Analysis

Level	Unsafe Control Actions and Combinations of Control Actions
System Level	UCA-6: MUM-T provides fix too late after the target is out of range UCA-10: MUM-T provides search after the object is out of range (H3) UCA-16: MUM-T provides search too late after mission window closes (H3)
Team Level	UCCA-15: MUM-T ends fix before it starts to fire when a target fired on must be fixed (H3, H5) UCCA-15.1: a UAS ends fix before TL starts to fire when… (same as above) UCCA-15.2: a UAS ends fix before it starts to fire when… (same as above) UCCA-15.3: a UAS ends fix before another UAS starts to fire when… (same)
Controller Level	UCA-28: UAS Autonomous Controller powers on payload before the conserve power option is no longer implemented UCA-38: UAS Autonomous Controller releases a missile after the target is out of range (H3, H5)

The causal scenarios developed for each of these UCAs and UCCAs are available in their respective analyses. Safety constraints can then be derived from them to include in the specification. Examples of safety constraint derived from the STPA analysis are provided in the baseline [53]. Safety constraints derived from the results of STPA-Teaming and their V&V are addressed in Section 6.3.

In addition to the conceptual architecture shown in Figure 6-2, different design alternatives can also be explored and analyzed using the same process. An example of this is in Chapter 5.6, where a new control channel is added to allow the UAS to task the TL to search (Figure 5-8). This interaction is analyzed at the team level and produces UCCAs, scenarios, and safety constraints that are traced to the design variation.

6.3 Design Guidance from Safety Constraints

Safety-guided design aims to proactively influence how a system is engineered throughout its developmental lifecycle so that it is built to be safe. STPA facilitates this by enabling the hazard analysis to start in early conceptual stages before the details of the design are known. The safety

constraints it produces are end-to-end traceable to the identified losses (Figure 6-3). These constraints are derived from the various STPA artifacts, including the hazards, the UCAs, and the loss scenarios. They inform how to design (or operate) the system to prevent these losses [50].

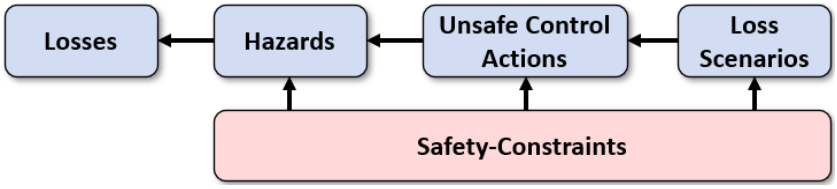


Figure 6-3. End-to-End Traceability of Safety constraints to Losses Using STPA

The framework developed in this chapter adopts this strategy for STPA-Teaming, but it also introduces a novel traceability concept to enhance safety-guided design. It establishes traceability between (1) the design decisions that shape how the control structure is modeled and (2) the safety constraints that are derived from the hazard analysis of that model. The intent is to provide direct insight into how certain design decisions influence the system safety requirements. This can influence whether or not those design decisions are implemented, and if so how.

The traceability of safety constraints to both losses and design decisions is summarized within the safety-guided framework in Figure 6-4. Examples from the MUM-T case study for how these paths are traced follow. A full set of safety constraints generated for MUM-T and traced using this approach is available in Appendix 5.

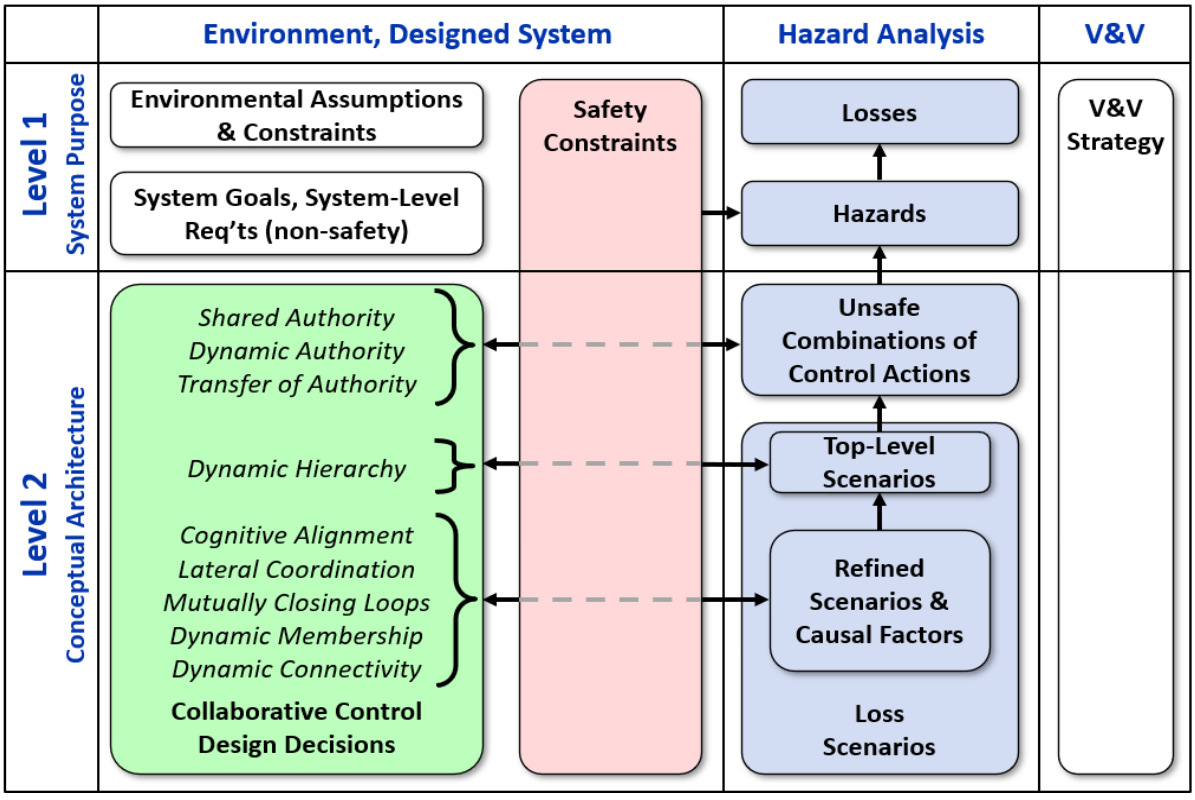


Figure 6-4. Traceability (Black Arrows) of Safety Constraints to Losses & Design Decisions

6.3.1 Tracing Safety constraints to Losses

All of the artifacts produced in STPA-Teaming are end-to-end traceable to the losses, as is also the case in STPA. The safety constraints derived from the hazards are the same as those considered in STPA. Those are generated at Level 1, as previously shown in Section 6.2.1.

At Level 2, where STPA-Teaming is executed in the case study, safety constraints can be derived directly from the UCCAs, the top-level scenarios, and the refined scenario causal factors. As in SPTA, these constraints inform how to implement the system design to avoid these items. For traceability, each safety constraint is tagged with the artifact(s) from which it is derived.

Table 6-2 illustrates how high-level constraints are derived from a UCCA and its top-level scenarios. Each safety constraint is generalized and is therefore applicable to multiple other UCCAs and their top-level scenarios, as listed in the traceability. The "[DA]" also included in the traceability is discussed next in Section 6.3.2.

Table 6-2. Example Safety constraints Derived from UCCA and Top-Level Scenarios

UCCA and Top-Level Scenarios	Derived Safety Constraints
UCCA 40.4: Ci does not fire and no other Cj fires when there is a priority target to engage and a teammate able to fix [H3] [DA]	**SC-2.1**: One of the controllers on the team must provide a control action if that action is necessary and the necessary resources to provide it are available. [UCCA 37.1, UCCA 40.4, UCCA 43.7, UCCA 1.1] [DA]
S-40.4.1: TL does not provide tasks necessary for the team to provide safe collective control. Here, TL does not task any UAS to fire (and does not fire her/himself).	**SC-2.1.1**: The team leadership must direct its teammate(s) to provide the control action if it does not plan to provide it itself. [S-37.1.1, S-40.4.1, S-43.7.1, S-1.1.1] [DA]
S-40.4.2: TL provides tasks to the team in a way that leads to unsafe collective control. Here, TL tasks multiple UAS to fire.	**SC-2.1.2**: The team leadership must not direct multiple teammates to provide a control action if multiple controllers providing that action results in an unsafe conflict. [S-37.1.2, S-3.1.2, S-40.4.2, S-42.6.2, S-1.1.2] [DA]
S-40.4.3: TL provides proper tasks to UAS, but some UAS do not execute them as provided. Here, TL tasks a UAS to fire, but it does not fire	**SC-2.1.3**: If a controller is properly directed by its team leadership to provide a control action, that controller must provide that action as directed. [S-37.1.3, S-40.4.3, S-1.1.3] [DA] **SC-2.1.3.1**: If a set of controllers as a whole is properly directed by the team leadership to provide a control action, then that set must allocate a controller to provide the action. [S-43.7.3, S-1.1.3] [DA]
S-40.4.4: *Not-Applicable to UCCA*	*Not-Applicable*
S-40.4.5: TL control actions to the process are unsafe with otherwise adequate tasks provided to the UAS. Here, TL does not intend to fire and does not task a UAS to fire.	**SC-2.1.5**: The team leadership must provide a control action itself if it is capable, that action is necessary, and it does not direct its teammate(s) to provide the action. [S-40.4.5, S-43.7.5, S-1.1.5] [DA]

6.3.2 Tracing Safety constraints to Design Decisions

The analytical structure of STPA-Teaming allows some of its artifacts to be related to the types of collaborative interactions explored. As was highlighted in Table 4-18, three of the collaborative dynamics are addressed in the identification of UCCAs, one other is found using top-level scenarios, and the remaining five are captured in the refinement of causal scenarios. As such, there is a link between the design decision to incorporate these interactions and the safety constraints, which is traced in Figure 6-4.

To enable this novel concept of traceability, the STPA-Teaming artifacts are labeled with the collaborative dynamic they relate to. The following convention is used.

CA: Cognitive Alignment	**DH**: Dynamic Hierarchy	**LC**: Lateral Coordination
DA: Dynamic Authority	**DM**: Dynamic Membership	**SA**: Shared Authority
DC: Dynamic Connectivity	**MC**: Mutually Closing Loops	**TA**: Transfer of Authority

Table 6-2 illustrates this concept. UCCA 40.4 reflects the decision to incorporate *dynamic authority* into the system, which results in the unsafe control gap identified by the UCCA. Therefore, the safety constraints derived from this UCCA and its scenarios are traceable to *dynamic authority* and are labeled with a "[DA]".

The scenario refinement process, introduced in Figure 4-21, systematically explores causal factors associated with five collaborative control dynamics. The safety constraints derived from the refined scenarios are therefore readily traceable to those types of interactions. For this reason, the safety constraints in the overall conceptual architecture specification can be clustered by, or searched for by, the collaborative interactions they are traced to. Some constraints may be traceable to multiple different collaborative control dynamics.

This grouping is demonstrated below using a partial set of the constraints traced to the *mutually-closing control loops* dynamic in the MUM-T case study. As in the example above, the general form of each constraint makes it traceable to multiple scenarios uncovered in the overall analysis. The traceability to both the loss and the collaborative dynamic is included for each constraint. Here, the indexing "SC-2.MC" means Level 2 Safety constraint found in Mutually Closing Control Loops.

SC-2.MC.1: A controller that relies on a teammate for feedback on its control action(s) to the shared process must adequately receive and accurately interpret that feedback. [MC]

- **SC-2.MC.1.1**: The controller must be able to interpret which control action the feedback from the teammate pertains to. [MC]
  - **SC-2.MC.1.1.1**: The controller must be able to interpret which controlled subprocess the feedback from the teammate pertains to. [S-37.1.3, S-40.4.3, S-40.4.5, S-1.1.3, S-1.1.5, S-4.4.3, S-4.4.5] [MC]
  - **SC-2.MC.1.1.2**: The controller must be able to interpret which controller the feedback from the teammate pertains to. [S-37.1.3, S-40.4.3, S-40.4.5, S-1.1.3, S-1.1.5, S-17.2.7] [MC]
- **SC-2.MC.1.2**: Feedback exchanged between two different types of controllers (e.g., a human controller and a machine controller) must adhere to a syntax that enables semantic alignment between the two. [S-37.1.3, S-38.2.3, S-40.4.3, S-40.4.5, S-1.1.3, S-1.1.5, S-3.3.3] [MC]

- **SC-2.MC.1.6**: The controller must be able to determine which teammate provided the feedback. [S-37.1.3, S-38.2.3, S-1.1.3] [MC]

- *More items in Appendix 5…*

SC-2.MC.2: A controller that relies on feedback from the process that is generated in response to a teammate's control action must be able to adequately receive and accurately interpret that feedback. [MC]

- **SC-2.MC.2.1**: The controller must verify that the feedback is in response to the expected control action provided by a teammate. [MC]
  - **SC-2.MC.2.1.1**: The controller must verify that the feedback is in response to the expected controlled subprocess. [S-40.4.3, S-40.4.5, S-1.1.5, S-4.4.3, S-4.4.5, S-15.1.7, S-15.1.8] [MC]
  - **SC-2.MC.2.1.2**: The controller must verify that the feedback is in response to the expected teammate. [S-37.1.3, 2.2.3, S-1.1.3, S-3.3.3, S-3.3.4, S-3.3.5, S-15.1.7, S-15.1.8] [MC]

- **SC-2.MC.2.2**: The controller and the teammate must be coordinated on when and how the teammate provides its control action so that the controller can adequately receive the feedback. [S-40.4.3, S-40.4.5] [MC]

- *More items in Appendix 5…*

The traceability offered in this framework enables designers to search for all the system-constraints associated with certain design decisions. The example above highlights some of the requirements necessary to implement a safe architecture where multiple controllers close each other's control loops. If those requirements are too difficult or costly to meet, designers may choose to eliminate the associated features altogether and simplify the architecture.

In the case of the MUM-T, the safety constraint examples above may guide an engineering decision to simplify the system by only allowing the same controller to provide both the fix and fire commands for a given target. This decision, guided by safety, eliminates many of the mutually closing control loop interactions and their design constraints. However, it may also limit the ability of the system to meet other measures of performance. Therefore, other design considerations beyond just safety must also be considered in the decision.

6.3.3 V&V of Safety Constraints

STPA and STPA-Teaming rigorously trace how the safety constraints are derived and why they are important. However, the constraints must still be validated and determined to be verifiable before they are formally incorporated into the design.

A variety of methods are available for this process. For example, one aircraft manufacturer conducts engineering and stakeholder reviews to determine the V&V maturity of requirements generated by STPA [218]. When necessary, this organization modifies the requirements to ensure their correctness while maintaining the rationale traced by the hazard analysis.

While designating a specific V&V approach is beyond the scope of this dissertation, the emphasis of the framework on tracing safety constraints to V&V is important to the integrated design-assurance paradigm. A properly V&V'd safety constraint provides a path to design safety

into the system and perform assurance by construction [4]. Similarly, an invalid or unverifiable safety constraint can inform design modification to avoid challenges in assurance.

6.4 Summary

The limitations in existing design and assurance processes, as reviewed in Chapter 2, have stifled the ability to field aerospace systems with complex team-inspired interactions. Existing techniques are founded on reductionism and do not scale well. Assurance is typically separated from design and considered too late in the engineering lifecycle. This chapter introduced a framework that aims to overcome some of these problems by integrating design and assurance activities using a system-theoretic approach.

The framework for safety-guided design is derived from Intent Specification [52], and it traces information across different design and assurance processes, levels of abstraction, and engineering views. The structure informs how to incorporate the extended hazard analysis technique, STPA-Teaming, into a broader systems engineering process.

As demonstrated in the MUM-T system case study, the framework integrates STPA and STPA-Teaming by shifting the focus of analysis to different interactions of interest. By selectively employing both techniques, their explanatory strengths complement one another to produce a more comprehensive analysis. The safety constraints produced by the analyses are rigorously traceable to the losses they are specified to prevent.

In addition to tracing safety constraints to losses, as done in STPA, the analytical structure of STPA-Teaming was leveraged to introduce novel traceability. The development of UCCAs and scenarios is grounded in the collaborative control dynamics defined in Chapter 3. As such, the safety constraints derived from these artifacts are also traceable to those dynamics. This empowers engineers to examine, with rigor, how the decisions to incorporate these complex interactions into the architecture affect the safety requirements.

As presented here, this new traceability concept focused on safety-guided design decisions to implement collaborative control interactions. However, the idea can be generalized to achieve other system emergent properties, such as security. The way in which the hazard analysis artifacts are connected to the conceptual architecture design decisions can also be expanded beyond collaborative control design decisions. The method to generalize this type of approach provides an opportunity to expand this research in future work.

Chapter 7: Conclusion and Future Work

The aerospace community is pursuing new system concepts that aim to improve how humans and machines, and multiple machines, work collaboratively. Many of their component interactions are inspired by the complex dynamics that occur in human teaming. Unfortunately, existing systems engineering processes are limited in their ability to model, analyze, design, and assure the safety of systems with such interactions. This gap has restricted the types of designs that can be fielded in safety-critical domains, and as such, few of the novel aerospace "teaming" systems have successfully been deployed.

The objective of this dissertation was to address part of this gap by developing a rigorous and systematic framework to analyze safety and perform safety-guided design of systems that exhibit collaborative control interactions. This objective was achieved through three technical contributions that advance the state-of-the-art in system safety engineering. The first is a mechanism to describe the different types of collaborative control interactions using the principles of Systems Theory. The second is an extended hazard analysis technique, named STPA-Teaming, which enhances the modeling and analysis of safety in collaborative control systems. The third is a framework that facilitates safety-guided design using the results of the extended hazard analysis process.

The methods developed in this work were demonstrated on a real-world aerospace system concept for manned-unmanned aircraft teaming (MUM-T). The MUM-T system was modeled in a way that explicitly captured its different collaborative control interactions. Using STPA-Teaming, new causal factors were identified that were not previously found in a past hazard analysis of the same system. Finally, the safety-guided design framework enabled direct traceability between the decision to incorporate different collaborative interactions into the MUM-T architecture and the safety requirements they impose on the design.

The following subsections summarize each of the three research contributions and review their limitations. They also suggest how, in many cases, these limitations open opportunities to expand this research in future work.

7.1 Contribution 1: Collaborative Control Definition

Chapter 3 introduced a system-theoretic framework to define collaborative control interactions so that they can be more completely analyzed using STAMP. It includes a taxonomy of seven dimensions to describe the structure of interactions between multiple controllers. These dimensions influence the causal relationships between controllers. They consider:

- The types of controllers: humans, machines, or a combination
- Hierarchal structure: from hierarchal control to peer interactions
- Behavioral intent: from cooperative to adversarial
- Connectivity: whether global, local, or disconnected

- Information exchange: active, passive, or none
- How roles & responsibilities are defined: prescribed, dynamic, ad-hoc
- Developmental origins: from co-designed to meeting in the field

The framework also defines nine dynamics observed in collaborative control systems. Their definitions are grounded in the principles of Systems Theory and the elements of STAMP. The dynamics include:

- Cognitive Alignment
- Lateral Coordination
- Mutually Closing Control Loops
- Shared Authority
- Dynamic Authority
- Transfer of Authority
- Dynamic Hierarchy
- Dynamic Membership
- Dynamic Connectivity

The framework creates the necessary foundation to extend system-theoretic hazard analysis methods to systematically explore causal factors associated with these collaborative relationships. It was evaluated over a set of 101 controller interactions found in aerospace systems described in the literature. The findings support *Hypothesis 1: the framework provides a mechanism to categorize and describe component interactions that are, or are planned to be, designed into aerospace systems*. The analysis also suggests that these complex collaborative dynamics are more prevalent in conceptual human-machine and multi-machine systems than in those already fielded.

There are several limitations recognized for this part of the work. First, no claim can be made that the set of dimensions in the taxonomy nor the set of collaborative control dynamics are complete. There are arguably other notable factors that influence how multiple controllers interact, especially in other domains beyond aerospace. There may also be other dynamics that are found to be important to account for in collaborative control.

Next, the evaluation of the framework over the set of controller interactions is limited in its quantitative power and is subject to author bias. The analyzed set of systems originated from the literature review and was expanded through interactions with subject matter experts from various technology domains. It was not a randomly selected sample, and the results are not necessarily representative of all aerospace systems.

Because the framework introduced in this work is new, the systems evaluated generally did not describe the component interactions using the same terms. The categorization of the interactions was based on the engineering judgment of the author, which involves subjectivity and bias. However, due diligence was employed to apply the framework as consistently as possible across the evaluated set.

These limitations offer opportunities to further develop the framework in future work. One exciting prospect is to leverage the way in which the taxonomy and the collaborative dynamics were developed to explore other broad types of control interactions beyond teaming. For example, this type of theoretical framework could be adapted to define and model competitive control interactions. The general approach of defining new interactions using system-theoretic principles to then specialize STPA for such relationships may help provide greater analytical depth in new research domains.

7.2 Contribution 2: STPA-Teaming Analytical Extensions

Chapter 4 extended the state-of-the-art for hazard analysis, STPA, to systematically address collaborative control. Three new techniques, collectively known as STPA-Teaming, were developed based on the foundation provided by the first contribution. STPA-Teaming aims to systematically analyze each of the nine collaborative control dynamics.

The first STPA-Teaming extension provides a mechanism to incorporate these dynamics into STAMP hierarchal control structure models. This ensures that the dynamics are consistently considered in hazard analysis. A *generic collaborative control structure* was expanded from the existing STAMP and STPA guidance to express how controllers collaborate to share control of a process. It serves as a template that can be reconfigured to model various human and/or machine team compositions.

The second extension establishes a process to identify *unsafe combinations of control actions* (UCCAs) to explore how the control actions of multiple teammates are unsafe together. It expanded the unsafe control action (UCA) structure defined in STPA to systematically consider potential unsafe gaps, overlaps, mismatches, and transfers of authority that are found in teams. An algorithm was developed to systematically abstract and refine the system model to linearize the combinatorial growth of UCCAs for arbitrarily complex systems. Its formalism enabled a tool to be created to automate the enumeration, refinement, pruning, and prioritization of UCCAs. This tool allows the human analyst to focus more on identifying the context in which the UCCAs are unsafe.

The third extension is a systematic approach to developing causal scenarios from the UCCAs. The process first defines scenarios at a high level, and then, iteratively refines them using a template that investigates various causal factors. These factors were derived from the STPA guidance that explores breakdowns in feedback control, and they were also framed by the collaborative control dynamics defined in the first contribution of this work.

Chapter 5 evaluated STPA-Teaming on the same MUM-T case study that was previously and independently analyzed using STPA. The extended technique was able to find new causal factors that were not considered in the previous analysis. The case study was then expanded to demonstrate how STPA-Teaming is able to handle all nine of the defined collaborative control dynamics.

The described rationale to extend STPA and the reviewed shortfalls of traditional hazard analysis techniques support *Hypothesis 2: the system-theoretic collaborative control framework describes component interactions that are not specifically addressed by existing hazard analysis techniques, including STPA.* The combined results of the case study support *Hypothesis 3: the hazard analysis extensions identify causal factors associated with collaborative control interactions, which are not systematically found using the existing STPA guidance.*

Several limitations of STPA-Teaming are acknowledged. First, the generic collaborative control structure is subject to the same limitations as in the first contribution. If new types of collaborative interactions were to be defined, they would need to be expressed in the collaborative control structure, and they may require some of the conventions introduced in this work to be modified to accommodate them.

Next, the limitations of the UCCA identification process were discussed in Chapter 4. As with any other linearization process, the approach used to manage combinatorial growth can only

approximate the different potential control combinations. While the approach provides a comprehensive and systematic search process using abstraction, certain refined combinations may be missed. Arbitrary expansions to the abstractions or full enumeration of all combinations can overcome this limitation.

The refined UCCA prioritization process in the last step of the algorithm relied on assumptions and engineering judgment. For this reason, the lower-priority UCCAs were retained in the output of the algorithm, so that they could be analyzed for scenarios if desired. Further work is necessary to specify a more general prioritization approach and to determine when lower-priority UCCAs can be skipped because of the diminishing return on new information their scenarios provide.

Finally, the scenario identification method cannot be claimed to be complete, as is also the case in baseline STPA. The structured approach that was developed is just one way to guide the process, and others may be conceived. Additional guidance may also be helpful to determine how far to refine each scenario. This also remains an open research question for baseline STPA.

There are many other opportunities to expand this method of analysis in future research. Because STPA-Teaming is nascent, it will benefit from lessons learned by applying it to additional case studies performed by other analysts. Some of this work is already underway as the technique is being employed by other graduate students for their research projects.

One prospect is to explore the conditions that allow STPA-Teaming to be simplified. The goal in this dissertation was to provide a comprehensive approach to analyze any collaborative system, no matter how complex. The solution was a procedure that is more complicated than STPA. Certain system conditions, beyond those specified in Section 4.2 regarding the applicability of the different abstractions, may help to reduce the types of control combinations that need to be considered. For example, some control actions may be shown to be independent of one another. Furthermore, the refinement of UCCAs may not always be necessary to generate causal scenarios. Practical approaches to reduce the complexity of the analysis will help expand the adoption of this technique.

Another opportunity is to expand the use of UCCAs beyond collaborative control interactions. New insights can be gained by exploring how multiple controllers acting on separate processes may be unsafe, as was initially studied by Placke using *conflict UCAs* [206]. The UCCA process provides a more comprehensive way to account for how multiple actions relate to each other. Similarly, the UCCA approach can uncover issues with one controller issuing combinations of multiple control actions. Further research is needed to understand when the benefits of the additional analytical complexity outweigh the costs.

7.3 Contribution 3: Safety-Guided Design Framework

Chapter 6 developed a framework that facilitates the safety-guided design of collaborative control systems. Its goal is to mitigate the problems encountered with existing practices, where safety assurance is typically carried out separately from design and too late in the engineering lifecycle. By integrating safety and assurance starting in early conceptual design stages, safety can be built

into the system from the beginning. It can also help ensure there is a feasible path to verification and validation (V&V) given the system complexity.

The approach, derived from Intent Specification, traces information across three dimensions of system development. First, the *design-assurance axis* aligns models of the system with their hazard analysis, derived safety constraints, and selected V&V strategies. Second, the *abstraction-refinement axis* supports the specification of the system at different levels of abstraction. For the collaborative systems emphasized in this work, the engineering focus shifts from (1) the interactions of a team as a whole, to (2) the interactions between the controllers within the team, and then (3) to lower-level component interactions for each controller. Third, the *engineering intent axis* describes the system using a means-end hierarchy, starting from concept of operation, down to lower-level component implementation.

The framework was demonstrated in the early engineering conceptual stages of the same MUM-T case study. This showed how to model the system at different levels of abstraction, and how to analyze those models with STPA and STPA-Teaming. A novel concept was also introduced to rigorously trace safety specifications derived from the analysis to conceptual architecture decisions, thereby enabling safety-guided design.

The scope of the research limited this portion of the work to a demonstration of how to integrate the first two contributions into a broader systems engineering framework. No formal evaluation of the correctness or performance of the framework was performed. Such an evaluation would require access to the safety engineering data for a collaborative control system developed using an alternate process, which was not available.

As with STPA-Teaming, the framework will benefit from future experiences applying it to additional system case studies and by more engineers. There are several other exciting opportunities for future research.

The first is to investigate how to generalize the novel approach to trace the safety constraints output from STPA-Teaming directly to conceptual architecture decisions. While this research focused on collaborative control, the same overall approach can be extended to link the results of the analysis directly to many other types of architectural decisions. The key is to determine how those decisions are addressed in the method of analysis and its artifacts.

Another important area to explore is how to more directly ensure the safety constraints derived from the hazard analysis are valid and verifiable. As described in Chapter 6, existing approaches do this iteratively by first specifying the constraints and then evaluating their maturity for V&V. Further research may be able to determine a way to specify safety constraints so that they meet V&V requirements upfront. This would help further couple design and assurance processes.

In addition, tools must be developed to facilitate traceability and operationalize this process for systems engineers. In this dissertation, automation was developed to support the identification of UCCAs. However, the traceability beyond that step was performed manually, which is intensive and error-prone.

Many commercial requirements management tools exist to link artifacts to one another, and some even support traceability in the baseline STPA technique [219]. Other tools are available to model designed systems using different systems engineering views similar to those on the vertical axis in the safety-guided framework [220]. These software systems illustrate that this

type of functionality is both feasible and in demand by professional engineers. However, the selection of a baseline tool and the implementation of the concepts introduced in this work is a substantial development effort itself that exceeded the scope of this research.

Finally, the analysis technique and the design framework may also be applied to achieve other emergent system properties beyond safety in collaborative control systems. Numerous studies have demonstrated how STAMP-based techniques can address cybersecurity [187] and other system behaviors. However, future work must fully specify how to do this so that it can be implemented in practice.

7.4 Final Note

This dissertation provides an avenue to support the analysis and design of systems with degrees of freedom in component interactions that go beyond those fielded today in aerospace. It alone does not solve all the engineering challenges associated with the development and deployment of such complex systems. Significant contributions from many different engineering domains are still required to ensure human-machine and multi-machine collaborative systems are safe.

This work does offers one approach to solicit and integrate contributions from these different technical disciplines to achieve safety. Opinions may differ on how to define, model, analyze, design, and assure systems with teaming interactions. However, the need for these systems to be safe when they become operational in society is incontrovertible. Regardless of how we get there, the author hopes that this dissertation inspires new ideas or empowers practitioners to engineer safety into teaming.

Acronyms

AAM: Advanced Air Mobility

AF: Air France

ACAS-X: Aircraft Collision Avoidance System

ARP: Aviation Recommended Practice

ATC: Air Traffic Control

ATM: Air Traffic Management

AuC: Autonomous Controller

AWACS: Airborne Warning And Control System

BCM: Behavior Competency Model

CA: Cognitive Alignment

CFR: Code of Federal Regulation

COA: Course of Action

ConOps: Concept of Operations

CTA: Critical Task Analysis

CRM: Crew Resource Management

CS: Causal Scenarios

CWA: Cognitive Work Analysis

DA: Dynamic Authority

DC: Dynamic Connectivity

DH: Dynamic Hierarchy

DM: Dynamic Membership

DoDAF: DoD Architecture Framework

DSM: Dependency Structure Matrix

EA: Environmental Assumption

EC: Environmental Constraint

EVS: Enhanced Vision System

FAA: Federal Aviation Administration

FHA: Functional Hazard Analysis

FM: Formal Methods

FMEA: Failure Modes and Effects Analysis

FMECA: Failure Modes and Effects Criticality Analysis

FRAM: Functional Resonance Analysis Method

FO: First Officer

FTA: Fault Tree Analysis

GAMA: General Aviation Manufacturing Association

GS: Ground Station

HAI: Human Automation Interaction

HAZOP: Hazards and Operability Analysis

HH: Human-Human Interaction

HM: Human-Machine Interaction

HMT: Human-Machine Teaming

HTA: Hierarchal Task Analysis

JTA: Job Training Analysis

LOA: Level of Automation

LC: Lateral Coordination

MABA-MABA: Men Are Better At – Machines Are Better At

MC: Mutually Closing Control Loops

MIT: Massachusetts Institute of Technology

ML: Machine Learning

MM: Machine-Machine Interaction

MTE: Mission Task Element

MUM-T: Manned-Unmanned Teaming

NAS: National Airspace System

NATO: North Atlantic Treaty Organization

OPD: Observe Predict Direct

RF: Radio Frequency

RTA: Run Time Assurance

RSO: Remote Supervisory Operations

SA: Shared Authority

SC: Safety Constraint

STAMP: System Theoretic Accident Model and Processes

STPA: System Theoretic Process Analysis

SVO: Simplified Vehicle Operation

SysML: System Modeling Language

TA: Transfer of Authority

TCAS: Traffic Collision Avoidance System

TL: Team Lead

UAM: Urban Air Mobility

UAS: Unmanned Aircraft System

UAV: Unmanned Air Vehicle

UCA: Unsafe Control Action

UCCA: Unsafe Combination of Control Actions

UxS: Unmanned System

V&V: Verification and Validation

Bibliography

[1] N. G. Leveson, “Safety III: A Systems Approach to Safety and Resilience,” 2020.

[2] M. Johnson, J. M. Bradshaw, P. J. Feltovich, C. M. Jonker, M. B. Van Riemsdijk, and M. Sierhuis, “Coactive Design: Designing Support for Interdependence in Joint Activity,” *J. Hum.-Robot Interact.*, vol. 3, no. 1, p. 43, Mar. 2014, doi: 10.5898/JHRI.3.1.Johnson.

[3] C. E. Billings, *Aviation automation: The search for a human-centered approach*. CRC Press, 2018.

[4] N. G. Leveson, *Introduction to System Safety Engineering*, In Production. Cambridge, MA: MIT Press, 2023.

[5] V. Battiste, J. Lachter, S. Brandt, A. Alvarez, T. Z. Strybel, and K.-P. L. Vu, “Human-Automation Teaming: Lessons Learned and Future Directions,” in *Human Interface and the Management of Information. Information in Applications and Services*, S. Yamamoto and H. Mori, Eds., Springer International Publishing, 2018, pp. 479–493.

[6] NASA and Deloitte, “Urban Air Mobility Concept of Operations (ConOps) - UAM Maturity Level (UML) 4.” NASA, Dec. 02, 2020. [Online]. Available: www.nasa.gov/aeroresearch/uam-vision-conops-uml-4

[7] Booz Allen Hamilton, “Urban Air Mobility (UAM) Market Study - Final Report Submitted to NASA,” Nov. 2018.

[8] J. Holbrook *et al.*, “Enabling Urban Air Mobility: Human-Autonomy Teaming Research Challenges and Recommendations,” presented at the AIAA AVIATION 2020 FORUM, Jun. 2020. doi: 10.2514/6.2020-3250.

[9] National Academies of Sciences, Engineering, and Medicine, *Advanced Aerial Mobility: A National Blueprint*. in Committee on Enhancing Air Mobility—A National Blueprint; Aeronautics and Space Engineering Board; Division on Engineering and Physical Sciences. Washington, D.C.: National Academies Press, 2020. doi: 10.17226/25646.

[10] GAMA, “A Rational Construct for Simplified Vehicle Operations (SVO).” General Aviation Manufacturers Association EPIC SVO Subcommittee Whitepaper, May 20, 2019.

[11] R. E. Bailey, L. J. Kramer, K. D. Kennedy, C. L. Stephens, and T. J. Etherington, “An assessment of reduced crew and single pilot operations in commercial transport aircraft operations,” presented at the 2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC), Sep. 2017, pp. 1–15. doi: 10.1109/DASC.2017.8101988.

[12] T. Pallini, “Your packages may soon be flown on a massive jet with only one pilot, and it’s only a matter of time before you could be too,” *Business Insider*, Dec. 05, 2021. Accessed: Jan. 19, 2022. [Online]. Available: www.businessinsider.com/airbus-cargo-plane-may-be-candidate-for-single-pilot-operations-2021-11

[13] J. Shively, “Tech Activity Update US (NASA) - Human Autonomy Teaming - Model, Agent, Principles & Patterns,” *NASA Ames Res. Cent.*, May 2017, [Online]. Available: ntrs.nasa.gov/citations/20170000308

[14] N. Ho *et al.*, “Application of human-autonomy teaming to an advanced ground station for reduced crew operations,” in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, St. Petersburg, FL: IEEE, Sep. 2017, pp. 1–4. doi: 10.1109/DASC.2017.8102124.

[15] “Next Generation Air Transportation System: Next Gen UAS Research, Development and Demonstration Roadmap.” NextGen Joint Planning and Development Office, Mar. 15, 2012. [Online]. Available: https://irp.fas.org/program/collect/uas-nextgen.pdf

[16] NASA, “NASA Technology Roadmaps TA 15: Aeronautics.” 2015. [Online]. Available: nasa.gov/sites/default/files/atoms/files/2015_nasa_technology_roadmaps_ta_15_aeronautics_final.pdf

[17] NASA, “NASA Aeronautics Strategic Implementation Plan.” 2017. [Online]. Available: www.nasa.gov/sites/default/files/atoms/files/sip-2017-03-23-17-high.pdf

[18] A. Kharsansky, “A systemic approach toward scalable, reliable and safe satellite constellations,” Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2020.

[19] T. Fong, “Human-robot teaming for space exploration.” NASA Intelligent Robotics Group, Oct. 11, 2017. [Online]. Available: ntrs.nasa.gov/api/citations/

[20] DoD, “Unmanned Systems Integrated Roadmap FY2011-2036.” Department of Defense, 2011. [Online]. Available: https://irp.fas.org/program/collect/usroadmap2011.pdf

[21] USAF, “Autonomous Horizons: System Autonomy in the Air Force - A Path to the Future. Volume I: Human-Autonomy Teaming.” United States Air Force Office of the Chief Scientist, 2015.

[22] A. M. Dropkin *et al.*, “Aircrew Labor In-Cockpit Automation System Flight Testing,” in *Aurora Flight Sciences*, p. 18.

[23] K. L. Mosier, U. Fischer, B. K. Burian, and J. A. Kochan, “Autonomous, Context-Sensitive, Task Management Systems and Decision Support Tools I: Human-Autonomy Teaming Fundamentals and State of the Art,” *NASA Rep.*, 2017, doi: 10.13140/RG.2.2.25859.60966.

[24] DoD, “Perdix Fact Sheet.” Department of Defense - Strategic Capabilities Office, 2017.

[25] A. Hudson, “The Looming Swarm,” *Air Force Magazine*, Mar. 22, 2019. www.airforcemag.com/article/the-looming-swarm/ (accessed Jan. 21, 2022).

[26] V. Insinna, “US Air Force completes tests of swarming munitions, but will they ever see battle?,” *Defense News*, Jun. 07, 2021. www.defensenews.com/air/2021/06/07/us-air-force-successfully-completes-tests-of-swarming-munitions-but-their-future-is-unclear/ (accessed Jan. 21, 2022).

[27] P. Calhoun, “Gremlins.” Defense Advanced Research Projects Agency, 2021. Accessed: Jan. 21, 2022. [Online]. Available: www.darpa.mil/program/gremlins

[28] A. Kopeikin *et al.*, “Rotary-Wing Aircraft Development Cybersecurity and Safety STPA Status Report.” MIT / MIT Lincoln Lab Research Report, Jan. 22, 2021.

[29] J. R. Fabijanowicz, “Design of Experiments for Air Launched Effects Unmanned Aerial Vehicles,” Masters Thesis, Naval Postgraduate School, Monterey, CA, 2020.

[30] Lockheed Martin, “HAVE RAIDER Demo - U.S. Air Force, Lockheed Martin Demonstrate Manned/Unmanned Teaming,” Feb. 06, 2018. www.lockheedmartin.com/en-us/capabilities/autonomous-unmanned-systems/unmanned-military-case-study-have-raider-demo.html (accessed Jan. 28, 2022).

[31] V. Insinna, “Under Skyborg program, F-35 and F-15EX jets could control drone sidekicks,” *Defense News*, May 22, 2019. www.defensenews.com/air/2019/05/22/under-skyborg-program-f-35-and-f-15ex-jets-could-control-drone-sidekicks/ (accessed Jan. 28, 2022).

[32] B. Clark, D. Patt, and H. Schramm, "Mosaic Warfare: Exploiting Artificial Intelligence and Autonomous Systems to Implement Decision-Centric Operations." Center for Strategic and Budgetary Assessments (CSBA), Feb. 11, 2020. [Online]. Available: https://csbaonline.org/research/publications/mosaic-warfare-exploiting-artificial-intelligence-and-autonomous-systems-to-implement-decision-centric-operations

[33] Airbus, "A Statistical Analysis of Commercial Aviation Accidents 1958-2016," 2017. Accessed: Feb. 01, 2022. [Online]. Available: https://accidentstats.airbus.com/

[34] D. A. Mindell, *Our Robots, Ourselves: Robotics and the Myths of Autonomy*. Penguin Publishing Group, 2015.

[35] "Final Report – On the Accident on 1st June 2009 to the Airbus A330-203 Registered F-GZCP Operated by Air France flight AF 447 Rio de Janeiro-Paris." Bureau D'Enquetes et d'Analyses pour la security de l'aviation civile, 2012.

[36] S. S. Silva and R. J. Hansman, "Divergence Between Flight Crew Mental Model and Aircraft System State in Auto-Throttle Mode Confusion Accident and Incident Cases," *J. Cogn. Eng. Decis. Mak.*, vol. 9, no. 4, pp. 312–328, Dec. 2015, doi: 10.1177/1555343415597344.

[37] P. Robinson, *Flying Blind - The 737 MAX Tragedy and the Fall of Boeing*. Doubleday, 2021.

[38] N. G. Leveson, *Engineering a safer world: systems thinking applied to safety*. in Engineering systems. Cambridge, MA: MIT Press, 2011.

[39] A. Pritchett, M. Portman, and T. Nolan, "Research & Technology Development for Human-Autonomy Teaming Final Report: Literature Review and Findings from Stakeholder Interviews," 2018.

[40] L. J. Prinzel, "Human-Autonomy Teaming - A Review of Literature and Preliminary Recommendations." NASA Langley Research Center, Transformational Tools and Technologies, Autonomous Systems, 2019.

[41] NATO HFM247, *Human-Autonomy Teaming: Supporting Dynamically Adjustable Collaboration*. Neuilly-sur-Seine: NATO, Research and Technology Organisation, 2020.

[42] M. M. Connors, "Concepts for the Design of Human-Autonomy Systems," *NASA Ames Res. Cent.*, 2017.

[43] K. Kearns, "DoD Autonomy Roadmap - Autonomy Community of Interest," presented at the NDIA 19th Annual Science & Engineering Technology Conference, Mar. 2018. [Online]. Available: https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2018/science/Kearns.pdf

[44] M. R. Endsley, "From Here to Autonomy: Lessons Learned From Human–Automation Research," *Hum. Factors*, vol. 59, no. 1, pp. 5–27, Feb. 2017, doi: 10.1177/0018720816681350.

[45] R. Parasuraman and V. Riley, "Humans and Automation: Use, Misuse, Disuse, Abuse," *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 39, no. 2, pp. 230–253, Jun. 1997, doi: 10.1518/001872097778543886.

[46] A. Dearden, M. Harrison, and P. Wright, "Allocation of function: scenarios, context and the economics of effort," *Int. J. Hum.-Comput. Stud.*, vol. 52, no. 2, pp. 289–318, Feb. 2000, doi: 10.1006/ijhc.1999.0290.

[47] National Research Council (U.S.) and Committee on Autonomous Vehicles in Support of Naval Operations, *Autonomous vehicles in support of naval operations*. Washington, D.C.: National Academies Press, 2005. Accessed: Jan. 22, 2022. [Online]. Available: http://public.eblib.com/choice/publicfullrecord.aspx?p=3377980

[48] E. E. Alves, D. Bhatt, B. Hall, K. Driscoll, A. Murugesan, and J. Rushby, "Considerations in Assuring Safety of Increasingly Autonomous Systems." NASA Report, 2018. [Online]. Available: https://ntrs.nasa.gov/api/citations/20180006312/downloads/20180006312.pdf

[49] Federal Aviation Administration, "14 CFR 21 - Certification Procedures for Products and Articles." 2022. [Online]. Available: ecfr.gov/current/title-14/part-21

[50] N. G. Leveson and J. P. Thomas, *STPA Handbook*. 2018. [Online]. Available: https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

[51] E. Salas, T. Dickinson, S. Converse, and S. Tannenbaum, "Toward an understanding of team performance and training," in *Teams: Their training and performance*, in Teams: Their training and performance. Westport, CT, US: Ablex Publishing, 1992, pp. xvi, 415.

[52] N. G. Leveson, "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, p. 21, 2000.

[53] J. Robertson, "System Theoretic Process Analysis Applied to Manned-Unmanned Teaming," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2019.

[54] J. Marschak, "Elements for a Theory of Teams," *Manag. Sci.*, vol. 1.2, pp. 127–137, 1955.

[55] D. R. Ilgen, "Teams embedded in organizations: Some implications," *Am. Psychol.*, vol. 54, no. 2, pp. 129–139, 1999, doi: 10.1037/0003-066X.54.2.129.

[56] Federal Aviation Administration, "Advisory Circular - AC 120-51E - Crew Resource Management Training." 2004. [Online]. Available: www.faa.gov/documentLibrary/media/Advisory_Circular/AC_120-51E.pdf

[57] V. Mancuso, "Lecture: Social Factors and Team Dynamics," presented at the 16.453 Human Systems Engineering, MIT, Cambridge, MA, Nov. 10, 2020.

[58] C. R. Paris, E. Salas, and J. A. Cannon-Bowers, "Teamwork in multi-person systems: a review and analysis," *Ergonomics*, vol. 43, no. 8, pp. 1052–1075, Aug. 2000, doi: 10.1080/00140130050084879.

[59] E. Salas, D. E. Sims, and C. S. Burke, "Is there a 'Big Five' in Teamwork?," *Small Group Res.*, vol. 36, no. 5, pp. 555–599, Oct. 2005, doi: 10.1177/1046496405277134.

[60] M. R. Endsley, "Toward a Theory of Situation Awareness in Dynamic Systems," *Hum. Factors*, vol. 37, no. 1, pp. 32–64, Mar. 1995, doi: 10.1518/001872095779049543.

[61] N. A. Stanton *et al.*, "Distributed situation awareness in dynamic systems: theoretical development and application of an ergonomics methodology," *Ergonomics*, vol. 49, no. 12–13, pp. 1288–1311, Oct. 2006, doi: 10.1080/00140130600612762.

[62] V. F. Mancuso, "Information Sciences and Technology," PhD Dissertation, Pennsylvania State University, State College, PA, 2012.

[63] S. Kozlowski and E. Salas, "A multilevel organizational systems approach for the implementation and transfer of training," in *Improving Training Effectiveness in Work Organizations*, K. Ford, Ed., Psychology Press, 2014, p. 287.

[64] A. Myne, "Introductions and Objectives," presented at the 2020 Human-Machine Teaming Technical Interchange Meeting, MIT Lincoln Lab, Lexington MA, Oct. 2020.

[65] J. Laird, C. Ranganath, and S. Gershman, "Future Directions in Human Machine Teaming Workshop," presented at the Future Directions Workshop Series Sponsored by Office of Under Secretary of Defense for Research and Engineering, Arlington, VA, 2019.

[66] P. Fitts, “Human Engineering for an Effective Air-Navigation and Traffic-Control System,” National Research Council, Washington, D.C., 1951. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADB815893.pdf

[67] J. E. Allen, C. I. Guinn, and E. Horvtz, “Mixed-initiative interaction,” *IEEE Intell. Syst. Their Appl.*, vol. 14, no. 5, pp. 14–23, Sep. 1999, doi: 10.1109/5254.796083.

[68] V. V. Unhelkar, S. Li, and J. A. Shah, “Decision-Making for Bidirectional Communication in Sequential Human-Robot Collaborative Tasks,” in *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, Cambridge United Kingdom: ACM, Mar. 2020, pp. 329–341. doi: 10.1145/3319502.3374779.

[69] G. Klein, D. D. Woods, J. M. Bradshaw, R. R. Hoffman, and P. J. Feltovich, “Ten challenges for making automation a ‘team player’ in joint human-agent activity,” *IEEE Intell. Syst.*, vol. 19, no. 6, pp. 91–95, Nov. 2004, doi: 10.1109/MIS.2004.74.

[70] D. B. Kaber, “Issues in Human–Automation Interaction Modeling: Presumptive Aspects of Frameworks of Types and Levels of Automation,” *J. Cogn. Eng. Decis. Mak.*, vol. 12, no. 1, pp. 7–24, Mar. 2018, doi: 10.1177/1555343417737203.

[71] A. R. Pritchett, “Aviation Automation: General Perspectives and Specific Guidance for the Design of Modes and Alerts,” *Rev. Hum. Factors Ergon.*, vol. 5, no. 1, pp. 82–113, Jun. 2009, doi: 10.1518/155723409X448026.

[72] H. C. Siu *et al.*, “Evaluation of Human-AI Teams for Learned and Rule-Based Agents in Hanabi,” *Adv. Neural Inf. Process. Syst.*, vol. 34, 2021.

[73] H. Mozannar, A. Satyanarayan, and D. Sontag, “Teaching Humans when to Defer to a Classifier via Exemplars,” *ArXiv211111297 Cs*, Dec. 2021, Accessed: Jan. 31, 2022. [Online]. Available: http://arxiv.org/abs/2111.11297

[74] B. M. Muir, “Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems,” *Ergonomics*, vol. 37, no. 11, pp. 1905–1922, Nov. 1994, doi: 10.1080/00140139408964957.

[75] B. M. Muir and N. Moray, “Trust in automation. Part II. Experimental studies of trust and human intervention in a process control simulation,” *Ergonomics*, vol. 39, no. 3, pp. 429–460, Mar. 1996, doi: 10.1080/00140139608964474.

[76] P. Cofta, *Trust, Complexity and Control*. Chichester, UK: John Wiley & Sons, Ltd, 2007. doi: 10.1002/9780470517857.

[77] J. D. Lee and K. A. See, “Trust in Automation: Designing for Appropriate Reliance,” *Hum. Factors*, vol. 46, no. 1, pp. 50–80, 2004.

[78] E. T. Chancey, M. S. Politowicz, and L. Le Vie, “Enabling Advanced Air Mobility Operations through Appropriate Trust in Human-Autonomy Teaming: Foundational Research Approaches and Applications,” presented at the AIAA Scitech 2021 Forum, Jan. 2021. doi: 10.2514/6.2021-0880.

[79] R. Murphey and P. M. Pardalos, *Cooperative Control and Optimization*, vol. 66. in Applied Optimization, vol. 66. Kluwer Academic Publishers, 2002.

[80] J. Ferber and G. Weiss, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, vol. 1. Reading: Addison-Wesley, 1999.

[81] G. Skorobogatov, C. Barrado, and E. Salamí, “Multiple UAV Systems: A Survey,” *Unmanned Syst.*, vol. 08, no. 02, pp. 149–169, Apr. 2020, doi: 10.1142/S2301385020500090.

[82] J. Hu *et al.*, "To Centralize or Not to Centralize: A Tale of Swarm Coordination," *ArXiv180501786 Cs*, May 2018, Accessed: Jan. 14, 2022. [Online]. Available: http://arxiv.org/abs/1805.01786

[83] L. B. Johnson, "Decentralized Task Allocation in Communication Contested Environments," PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2016.

[84] W. Ren and R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*. in Communications and Control Engineering. London: Springer London, 2008. doi: 10.1007/978-1-84800-015-5.

[85] A. Kopeikin, S. S. Ponda, and J. P. How, "Control of Communication Networks for Teams of UAVs," in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos, Eds., Dordrecht: Springer Netherlands, 2015. doi: 10.1007/978-90-481-9707-1.

[86] A. N. Kopeikin, "Dynamic Mission Planning for Communication Control in Multiple Unmanned Aircraft Teams," Massachusetts Institute of Technology, Cambridge, MA, 2012.

[87] R. Olfati-Saber and R. M. Murray, "Consensus Problems in Networks of Agents With Switching Topology and Time-Delays," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1520–1533, Sep. 2004, doi: 10.1109/TAC.2004.834113.

[88] W. Yu, Ed., *Distributed cooperative control of multi-agent systems*. Singapore: Wiley, 2017.

[89] C. Heisey, M. Brittain, D. Maki, and K. Bush, "Multi-Agent Systems Collaborative Teaming (MASCOT) Definition Process to Create Specifications for Multi-Agent System (MAS) Development," Nov. 2020.

[90] J. D. Schierman, M. D. DeVore, N. D. Richards, and M. A. Clark, "Runtime Assurance for Autonomous Aerospace Systems," *J. Guid. Control Dyn.*, vol. 43, no. 12, pp. 2205–2217, Dec. 2020, doi: 10.2514/1.G004862.

[91] E. Ordoukhanian and A. Madni, "Model-Based Approach to Engineering Resilience in Multi-UAV Systems," *Systems*, vol. 7, no. 1, Feb. 2019, doi: 10.3390/systems7010011.

[92] E. Crawley *et al.*, "The Influence of Architecture in Engineering Systems," in *Engineering Systems Monograph*, Cambridge, MA, 2004.

[93] L. F. Osborne, J. Brummond, R. Hart, M. Zarean, S. M. Conger, and Inc. Iteris, "Clarus: Concept of Operations," Federal Highway Administration, US Department of Transportation, FHWA-JPO-05-072, Oct. 2005. Accessed: Feb. 01, 2022. [Online]. Available: https://rosap.ntl.bts.gov/view/dot/3710

[94] INCOSE, *Systems Engineering Handbook - A Guide for System Life Cycle Processes and Activities*, INCOSE-TP-2003-002-03. International Council on Systems Engineering (INCOSE), 2006.

[95] E. Crawley, "16.842 Fundamentals of Systems Engineering - Session 3: Creating Value Through System Thinking," Massachusetts Institute of Technology, Cambridge MA, Sep. 15, 2020.

[96] E. Crawley, "16.842 Fundamentals of Systems Engineering - Session 4: System Architecture and Concept Generation," Massachusetts Institute of Technology, Cambridge MA, Sep. 22, 2020.

[97] L. Delligatti, *SysML Distilled - A Brief Guid to the System Modeling Language*. Addison-Wesly, 2014.

[98] E. Crawley, B. Cameron, and D. Selva, *System Architecture - Strategy and Product Development for Complex Systems*. Pearson, 2016.

[99] R. P. Bhattacharyya and A. R. Pritchett, "Designing Function Allocations in Air Traffic Concepts of Operation Using Network Optimization," *J. Air Transp.*, vol. 25, no. 2, pp. 61–72, Apr. 2017, doi: 10.2514/1.D0075.

[100] "DoD Architecture Framework V2.02," ASD(NII)/DoD CIO, Washington, D.C., 2011.

[101] N. Leveson, "An Improved Design Process for Complex, Control-Based Systems Using STPA and a Conceptual Architecture," Cambridge, MA, White Paper, 2020. [Online]. Available: http://sunnyday.mit.edu/conceptual-architecture-final.pdf

[102] J. Annett, "Hierarchical Task Analysis," in *Handbook of cognitive task design 2*, 2003, pp. 17–35.

[103] G. H. Walker, H. Gibson, N. A. Stanton, C. Baber, P. Salmon, and D. Green, "Event analysis of systemic teamwork (EAST): a novel integration of ergonomics methods to analyse C4i activity," *Ergonomics*, vol. 49, no. 12–13, pp. 1345–1369, Oct. 2006, doi: 10.1080/00140130600612846.

[104] Department of Defense, "MIL-STD-46855A – Human Engineering Requirements for Military Systems, Equipment, and Facilities." DoD Standard Practice, 2011.

[105] R. R. Copeland, "An Analysis and Classification Process towards the Qualification of Autonomous Functions in Army Aviation," presented at the Vertical Flight Society's 75th Annual Forum & Technology Display, Philadelphia PA, 2019.

[106] GTRAC, "Degraded Visual Environment (DVE) Critical Task Analysis - Task 2 Final Report," Georgia Tech Applied Research Corporation, Atlanta, GA, Oct. 2020.

[107] Federal Aviation Administration, "Advisory Circular - AC 120-54A - Crew Resource Management Training." 2017. [Online]. Available: www.faa.gov/documentLibrary/media/Advisory_Circular/AC_120-54A.pdf

[108] T. M. Longridge, "Overview of the Advanced Qualification Program," *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 41, no. 2, pp. 898–901, Oct. 1997, doi: 10.1177/107118139704100240.

[109] M. Feary, "A First Look at the Evolution of Flight Crew Requirements," p. 11.

[110] K. Wasson, N. Neogi, M. Graydon, J. Maddalon, P. Miner, and G. F. McCormick, "Functional Hazard Assessment for the eVTOL Aircraft Supporting Urban Air Mobility (UAM) Applications: Exploratory Demonstrations," *NASA Rep.*, p. 68, 2020.

[111] Administrative Committee 377 (AC377), "ASTM TR-1: Autonomy Design and Operations in Aviation: Terminology and Requirements Framework." ASTM International, Jan. 01, 2019. doi: 10.1520/TR1-EB.

[112] B. Lascara, A. Lacher, M. DeGarmo, L. Vempati, and R. Zimmerman, "Behavioral Competency Model for Safety Assurance of Automated Aviation Systems," in *AIAA Aviation 2019 Forum*, Dallas, Texas: American Institute of Aeronautics and Astronautics, Jun. 2019. doi: 10.2514/6.2019-3256.

[113] D. Wing, E. Chancey, M. Politowicz, and M. Ballin, *Achieving Resilient In-Flight Performance for Advanced Air Mobility through Simplified Vehicle Operations*. 2020. doi: 10.2514/6.2020-2915.

[114] E. Hollnagel, "RAG-The resilience analysis grid," in *Resilience engineering in practice - A guidebook*, Farnham, UK: Ashgate, 2011, pp. 275–296.

[115] K. J. Vincente, *Cognitive work analysis: Toward safe, productive, and healthy computer-based work*. CRC Press, 1999.

[116] A. R. Pritchett, S. Y. Kim, and K. M. Feigh, "Modeling Human–Automation Function Allocation," *J. Cogn. Eng. Decis. Mak.*, vol. 8, no. 1, pp. 33–51, Mar. 2014, doi: 10.1177/1555343413490944.

[117] J. Rasmussen, A. M. Pejtersen, and L. P. Goodstein, *Cognitive systems engineering*. in Wiley series in systems engineering. New York: Wiley, 1994.

[118] K. M. Feigh and A. R. Pritchett, "Requirements for Effective Function Allocation: A Critical Review," *J. Cogn. Eng. Decis. Mak.*, vol. 8, no. 1, pp. 23–32, Mar. 2014, doi: 10.1177/1555343413490945.

[119] R. J. Crouser and R. Chang, "An Affordance-Based Framework for Human Computation and Human-Computer Collaboration," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2859–2868, Dec. 2012, doi: 10.1109/TVCG.2012.195.

[120] S. W. A. Dekker and D. D. Woods, "MABA-MABA or Abracadabra? Progress on Human-Automation Co-ordination," *Cogn. Technol. Work*, vol. 4, no. 4, pp. 240–244, Nov. 2002, doi: 10.1007/s101110200022.

[121] R. J. Crouser, A. Ottley, and R. Chang, "Balancing Human and Machine Contributions in Human Computation Systems," in *Handbook of Human Computation*, P. Michelucci, Ed., New York, NY: Springer New York, 2013, pp. 615–623. doi: 10.1007/978-1-4614-8806-4_48.

[122] T. B. Sheridan and W. L. Verplank, *Human and computer control of undersea teleoperators*. Massachusetts Institute of Technology, Cambridge Man-Machine Systems Lab, 1978.

[123] M. R. Endsley and D. B. Kaber, "Level of automation effects on performance, situation awareness and workload in a dynamic control task," *Ergonomics*, vol. 42, no. 3, pp. 462–492, Mar. 1999, doi: 10.1080/001401399185595.

[124] SAE, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems," *Soc. Automot. Eng. -Road Autom. Veh. Stand. Comm.*, vol. SAE Standard J 3016, 2014.

[125] A. R. Pritchett, K. M. Feigh, S. Y. Kim, and S. K. Kannan, "Work Models that Compute to Describe Multiagent Concepts of Operation: Part 1," *J. Aerosp. Inf. Syst.*, vol. 11, no. 10, pp. 610–622, Oct. 2014, doi: 10.2514/1.I010146.

[126] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *IEEE Trans. Syst. Man Cybern. - Part Syst. Hum.*, vol. 30, no. 3, pp. 286–297, May 2000, doi: 10.1109/3468.844354.

[127] C. A. Miller and R. Parasuraman, "Designing for Flexible Interaction Between Humans and Automation: Delegation Interfaces for Supervisory Control," *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 49, no. 1, pp. 57–75, Feb. 2007, doi: 10.1518/001872007779598037.

[128] C. Ericson, *Hazard Analysis Techniques for System Safety*. John Wiley & Sons, Ltd, 2005.

[129] USAF, "Air Force System Safety Handbook." Air Force Safety Agency, Kirtland AFB NM, 2000.

[130] G. Weinberg, *An Introduction to General Systems Thinking*. Dorcet House Publishing, New York, NY, 2001.

[131] SAE, "ARP-4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment." Society of Automotive Engineers, SAE ARP4761, SAE International, Warrendale, PA, 1996.

[132] FAA, "Advisory Circular AC 23.1309: System Safety Analysis and Assessment for Part 23 Airplanes." Department of Transportation - Federal Aviation Administration, 2011.

[133] Department of Defense, "MIL-STD-882E - System Safety." May 11, 2012.

[134] P. Spirtes, C. N. Glymour, R. Scheines, and D. Heckerman, *Causation, prediction, and search*. MIT Press, 2000.

[135] D. M. Hausman, "Review Article: The Mathematical Theory of Causation," *Br. J. Philos. Sci.*, vol. 50, no. 1, pp. 151–162, 1999.

[136] N. G. Leveson and J. Thomas, "16-858 Introduction to Discrete Mathematics and System Theory - Lecture Notes," Massachusetts Institute of Technology, Cambridge MA, 2021.

[137] RTCA, "DO-178C: Software Considerations in Airborne Systems and Equipment Certification." Radio Technical Commission for Aeronautics, Washington DC, 2011.

[138] Standard Committee SC-147, "DO-385: Test Suite Encounter Set for Airborne Collision Avoidance System X (ACAS X) (ACAS Xa and ACAS Xo)." RTCA Inc., Washington DC, 2018.

[139] M. Graydon, N. A. Neogi, and K. Wasson, "Guidance for Designing Safety into Urban Air Mobility: Hazard Analysis Techniques," in *AIAA Scitech 2020 Forum*, Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 2020. doi: 10.2514/6.2020-2099.

[140] C. Belcastro, R. Newman, J. Evans, D. Klyde, L. Barr, and E. Ancel, *Hazards Identification and Analysis for Unmanned Aircraft System Operations*. 2017. doi: 10.2514/6.2017-3269.

[141] A. A. Baig, R. Ruzli, and A. B. Buang, "Reliability Analysis Using Fault Tree Analysis: A Review," *Int. J. Chem. Eng. Appl.*, pp. 169–173, 2013, doi: 10.7763/IJCEA.2013.V4.287.

[142] S. J. Levulis, P. R. DeLucia, and S. Y. Kim, "Effects of Touch, Voice, and Multimodal Input, and Task Load on Multiple-UAV Monitoring Performance During Simulated Manned-Unmanned Teaming in a Military Helicopter," *Hum. Factors*, vol. 60, no. 8, pp. 1117–1129, Dec. 2018, doi: 10.1177/0018720818788995.

[143] E. B. Johnson, A. N. Kopeikin, N. G. Leveson, and A. W. Drysdale, "Hazard Analysis for Human Supervisory Control of Multiple Unmanned Aircraft Systems," presented at the 56th International Symposium on Aviation Psychology, 2021, p. 274.

[144] N. G. Leveson, "16-355 Systems Engineering for Software Intensive Systems - Lecture Notes," Massachusetts Institute of Technology, Cambridge MA, 2021.

[145] A. Kopeikin, A. Clare, O. Toupet, J. How, and M. Cummings, *Flight Testing a Heterogeneous Multi-UAV System with Human Supervision*. 2012. doi: 10.2514/6.2012-4825.

[146] A. N. Kopeikin, S. S. Ponda, L. B. Johnson, and J. P. How, "Dynamic Mission Planning for Communication Control in Multiple Unmanned Aircraft Teams," *Unmanned Syst.*, vol. 01, no. 01, pp. 41–58, Jul. 2013, doi: 10.1142/S2301385013500039.

[147] "New generation of drones set to revolutionize warfare," *CBS 60 Minutes*, Jan. 21, 2017. [Online]. Available: https://www.cbsnews.com/news/60-minutes-autonomous-drones-set-to-revolutionize-military-technology/

[148] A. Kopeikin, S. Heider, D. Larkin, C. Korpela, R. Morales, and J. E. Bluman, "Unmanned Aircraft System Swarm for Radiological and Imagery Data Collection," in *AIAA Scitech 2019 Forum*, American Institute of Aeronautics and Astronautics, 2019. doi: 10.2514/6.2019-2286.

[149] A. Kopeikin *et al.*, "Designing and Flight-Testing a Swarm of Small UAS to Assist Post-Nuclear Blast Forensics," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, Sep. 2020, pp. 466–472. doi: 10.1109/ICUAS48674.2020.9213863.

[150] S. Mitra, *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. MIT Press, 2021.

[151] RTCA, "DO-333: Formal methods supplement to DO-178C." Radio Technical Commission for Aeronautics, Washington DC, 2011.

[152] F38 Committee, "ASTM F3269-17: Standard Practice for Methods to Safely Bound Flight Behavior of Unmanned Aircraft Systems Containing Complex Functions," ASTM International. doi: 10.1520/F3269-17.

[153] D. Cofer and S. Miller, "DO-333 Certification Case Studies," in *NASA Formal Methods*, in Lecture Notes in Computer Science, vol. 8430. Cham: Springer International Publishing, 2014, pp. 1–15. doi: 10.1007/978-3-319-06200-6_1.

[154] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," *IEEE Softw.*, vol. 30, no. 3, pp. 50–57, May 2013, doi: 10.1109/MS.2013.43.

[155] B. Weyers, J. Bowen, A. Dix, and P. Palanque, Eds., *The Handbook of Formal Methods in Human-Computer Interaction*. in Human–Computer Interaction Series. Cham: Springer International Publishing, 2017. doi: 10.1007/978-3-319-51838-1.

[156] A. Degani and M. Heymann, "Formal Verification of Human-Automation Interaction," *Hum. Factors*, vol. 44, no. 1, pp. 28–43, Mar. 2002, doi: 10.1518/0018720024494838.

[157] M. L. Bolton, "Novel Developments in Formal Methods for Human Factors Engineering," *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 61, no. 1, pp. 715–717, Sep. 2017, doi: 10.1177/1541931213601664.

[158] M. L. Bolton, E. J. Bass, and R. I. Siminiceanu, "Using Formal Verification to Evaluate Human-Automation Interaction: A Review," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 43, no. 3, pp. 488–503, May 2013, doi: 10.1109/TSMCA.2012.2210406.

[159] A. J. Abbate and E. J. Bass, "Modeling Affordance Using Formal Methods," *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 61, no. 1, pp. 723–727, Sep. 2017, doi: 10.1177/1541931213601666.

[160] M. Cubuktepe and U. Topcu, "Intent Prediction in Shared Control with Delayed Feedback," *Proc. Hum. Factors Ergon. Soc. Annu. Meet.*, vol. 61, no. 1, pp. 733–734, Sep. 2017, doi: 10.1177/1541931213601668.

[161] F. Vicentini, M. Askarpour, M. G. Rossi, and D. Mandrioli, "Safety Assessment of Collaborative Robotics Through Automated Formal Verification," *IEEE Trans. Robot.*, vol. 36, no. 1, pp. 42–61, Feb. 2020, doi: 10.1109/TRO.2019.2937471.

[162] E. J. Bass *et al.*, "Toward a multi-method approach to formalizing human-automation interaction and human-human communications," in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, Anchorage, AK, USA: IEEE, Oct. 2011, pp. 1817–1824. doi: 10.1109/ICSMC.2011.6083935.

[163] C. Muñoz and A. Narkawicz, "Formal Analysis of Extended Well-Clear Boundaries for Unmanned Aircraft," in *NASA Formal Methods*, S. Rayadurgam and O. Tkachuk, Eds., in Lecture Notes in Computer Science, vol. 9690. Cham: Springer International Publishing, 2016, pp. 221–226. doi: 10.1007/978-3-319-40648-0_17.

[164] A. Narkawicz, C. Munoz, and A. Dutle, “Sensor Uncertainty Mitigation and Dynamic Well Clear Volumes in DAIDALUS,” in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, London: IEEE, Sep. 2018, pp. 1–8. doi: 10.1109/DASC.2018.8569468.

[165] A. M. Dutle, C. A. Muñoz, A. J. Narkawicz, and R. W. Butler, “Software Validation via Model Animation,” in *Tests and Proofs*, J. C. Blanchette and N. Kosmatov, Eds., in Lecture Notes in Computer Science, vol. 9154. Cham: Springer International Publishing, 2015, pp. 92–108. doi: 10.1007/978-3-319-21215-9_6.

[166] V. A. Carreño, “Evaluation, Analysis and Results of the DANTi Flight Test Data, the DAIDALUS Detect and Avoid Algorithm, and the DANTi Concept for Detect and Avoid in the Cockpit,” *NASA Rep.*, p. 65, 2020.

[167] I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M. Viswanathan, “Runtime Assurance Based On Formal Specifications,” *Dep. Pap. CIS*, p. 14, 1999.

[168] S. D. Young, C. Quach, K. Goebel, and J. Nowinski, “In- Time Safety Assurance Systems for Emerging Autonomous Flight Operations,” in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, Sep. 2018, pp. 1–10. doi: 10.1109/DASC.2018.8569689.

[169] D. E. Swihart *et al.*, “Automatic Ground Collision Avoidance System design, integration, & flight test,” *IEEE Aerosp. Electron. Syst. Mag.*, vol. 26, no. 5, pp. 4–11, May 2011, doi: 10.1109/MAES.2011.5871385.

[170] J. M. Wing, “A specifier’s introduction to formal methods,” *Computer*, vol. 23, no. 9, pp. 8–22, Sep. 1990, doi: 10.1109/2.58215.

[171] R. A. De Millo, R. J. Lipton, and A. J. Perlis, “Social processes and proofs of theorems and programs,” *Commun. ACM*, vol. 22, no. 5, pp. 271–280, May 1979, doi: 10.1145/359104.359106.

[172] C. A. R. Hoare, “An axiomatic basis for computer programming,” *Commun. ACM*, vol. 12, no. 10, pp. 576–580, Oct. 1969, doi: 10.1145/363235.363259.

[173] US Army, “Army Regulation AR 70-62: Airworthiness of Aircraft Systems.” May 11, 2016.

[174] N. G. Leveson, “White Paper on the Use of Safety Cases in Certification and Regulation.” 2012. [Online]. Available: http://sunnyday.mit.edu/SafetyCases.pdf

[175] N. G. Leveson, “White Paper on Limitations of Safety Assurance and Goal Structuring Notation (GSN).” 2012. [Online]. Available: http://sunnyday.mit.edu/safety-assurance.pdf

[176] S. Cook, A. Dietrich, L. Hook, and A. Lacher, “Promoting Autonomy Design and Operations in Aviation,” in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, Sep. 2019, pp. 1–9. doi: 10.1109/DASC43569.2019.9081809.

[177] S. Cook, A. Dietrich, L. Hook, W. Ryan, and D. M. Stevens, “Advancing Autonomy in Aviation: A Holistic Approach,” in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, San Antonio, TX, USA: IEEE, Oct. 2020, pp. 1–8. doi: 10.1109/DASC50938.2020.9256568.

[178] FAA, “Certificates of Waiver or Authorization (COA).” https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/systemops/aaim/organizations/uas/coa

[179] M. Aust, E. Pennington, and W. Young, “STPA in Agility Prime,” presented at the STAMP Conference, Massachusetts Institute of Technology, Cambridge MA, Jun. 2021. [Online]. Available: http://psas.scripts.mit.edu/home/wp-content/uploads/2021/06/2021-06-29-1210__Aust.pdf

[180] "System Wide Safety Project – Assuring Increasingly Autonomous Systems with Non-Traditional Human Machine Roles," NASA Langley Research Center, Oct. 05, 2020.

[181] N. Leveson, "Healthcare Safety Research Grant Proposal." 2022.

[182] N. Leveson, "The Drawbacks in Using The Term 'System of Systems,'" *Biomed. Instrum. Technol.*, vol. 47, no. 2, pp. 115–118, Apr. 2013, doi: 10.2345/0899-8205-47.2.115.

[183] P. Checkland, *Systems Thinking, Systems Practice*. Wiley, 1999.

[184] Y. Zhang, C. Dong, W. Guo, J. Dai, and Z. Zhao, "Systems theoretic accident model and process (STAMP): A literature review," *Saf. Sci.*, p. 105596, Nov. 2021, doi: 10.1016/j.ssci.2021.105596.

[185] J. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Massachusetts Institute of Technology, Cambridge MA: McGraw-Hill Companies Inc., 2000.

[186] C. F. Kurtz and D. J. Snowden, "The new dynamics of strategy: Sense-making in a complex and complicated world," *IBM Syst. J.*, vol. 42, p. 22, 2003.

[187] W. E. Young, "Systems-Theoretic Security Engineering Analysis," PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2016.

[188] D. R. Montes, "Using STPA to inform developmental product testing," PhD Dissertation, Massachusetts Institute of Technology, 2016.

[189] M. E. France, "STPA - Engineering for Humans," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2017.

[190] D. S. Castilho, "Active STPA: Integration of Hazard Analysis into a Safety Management System Framework," PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2019.

[191] K. E. Johnson, "Systems-Theoretic Safety Analyses Extended for Coordination," PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2017.

[192] D. C. Horney, "Systems-Theoretic Process Analysis and Safety-Guided Design of Military Systems," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2015.

[193] B. R. Abrecht, "Systems Theoretic Process Analysis Applied to an Offshore Supply Vessel Dynamic Positioning System," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2016.

[194] J. M. Mackovjak, "Systems Theoretic Accident Analysis of an Offshore Supply Vessel Collision," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2016.

[195] N. A. Peper, "Systems Thinking Applied to Automation and Workplace Safety," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2017.

[196] B. Wong, "A STAMP Model of the Oberlingen Aircraft Collision Accident," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2004.

[197] C. Perrow, *Normal Accidents: Living with High Risk Technologies*. Princeton University Press, 1984.

[198] T. A. Saurin and R. Patriarca, "A taxonomy of interactions in socio-technical systems: A functional perspective," *Appl. Ergon.*, vol. 82, p. 102980, Jan. 2020, doi: 10.1016/j.apergo.2019.102980.

[199] H. V. D. Parunak, S. Brueckner, M. Fleischner, and J. J. Odell, "A Design Taxonomy of Multi-Agent Interactions," in *Agent-oriented software engineering IV: 4th international workshop, AOSE.*, P. Giorgini, J. P. Müller, and J. J. Odell, Eds., New York: Springer, 2004, pp. 132–146.

[200] A. Espinosa, J. Lerch, and R. Kraut, "Explicit vs. Implicit Coordination Mechanisms and Task Dependencies: One Size Does Not Fit All," in *Team Cognition: Process and Performance at the Inter- and Intro-Indivdual Level*, 2002.

[201] J. Leplat, "Occupational accident research and systems approach," *J. Occup. Accid.*, vol. 6, no. 1–3, pp. 77–89, Sep. 1984, doi: 10.1016/0376-6349(84)90036-1.

[202] SC-147, "DO-386: Minimum Operational Performance Standards for Airborne Collision Avoidance System Xu (ACAS Xu)." RTCA Inc., Washington DC, 2020.

[203] J. Thomas, L. Leveson Nancy G., N. Ishimama, M. Katahira, N. Hoshino, and K. Kakimoto, "STAMP Accident Model of HITOMI and Expansion to Future Safety Culture," presented at the STAMP Workshop, Massachusetts Institute of Technology, Cambridge MA, 2017. [Online]. Available: http://psas.scripts.mit.edu/home/wp-content/uploads/2017/04/Thomas-A-Process-for-STPA.pdf

[204] S. Bharadwaj, S. Carr, N. Neogi, and U. Topcu, "Decentralized Control Synthesis for Air Traffic Management in Urban Air Mobility," *IEEE Trans. Control Netw. Syst.*, vol. 8, no. 2, pp. 598–608, Jun. 2021, doi: 10.1109/TCNS.2021.3059847.

[205] J. Thomas, "Extending and automating a Systems-Theoretic hazard analysis for requirements generation and analysis.," PhD Dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2012. doi: 10.2172/1044959.

[206] M. S. Placke, "Application of STPA to the Integration of Multipole Control Systems: A Case Study and New Approach," Masters Thesis, Massachusetts Institute of Technology, Cambridge, MA, 2014.

[207] C. Fan and J. Deshmukh, "16.S398 Formal Methods for Safe Autonomous Systems - Lecture on Specifications: Temporal Logics." Apr. 2021.

[208] J. P. Thomas, "A New Process for Building STPA Causal Scenarios," presented at the STAMP Workshop, Massachusetts Institute of Technology, Cambridge MA, 2016.

[209] J. P. Thomas, "Empirical Evaluations of STPA in the Aviation Industry," presented at the STAMP Workshop, Massachusetts Institute of Technology, Cambridge MA, 2023.

[210] J. Thomas, "Enhancing Human Factors Analysis with STPA," presented at the STAMP Workshop, Massachusetts Institute of Technology, Cambridge MA, 2021.

[211] J. P. How, "16.31: Feedback Control Systems - Lecture Notes," Massachusetts Institute of Technology, Cambridge MA, 2011.

[212] Federal Aviation Administration, "14 CFR 91 - General Operating and Flight Rules." 2022. [Online]. Available: www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-91

[213] K. L. Hobbs, C. Cargal, E. Feron, and R. S. Burns, "Early Safety Analysis of Manned-Unmanned Team System," in *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, Kissimmee, Florida: American Institute of Aeronautics and Astronautics, Jan. 2018. doi: 10.2514/6.2018-1984.

[214] K. L. Hobbs *et al.*, "Systems Theoretic Process Analysis of a Run Time Assured Neural Network Control System." arXiv, Sep. 01, 2022. [Online]. Available: http://arxiv.org/abs/2209.00552

[215] IEEE, “ISO/IEC/IEEE 29148: Systems and Software Engineering - Life Cycle Processes - Requirements Engineering.” 2018.

[216] DoD, “DoD Instruction 5000.85: Major Capability Acquisition.” Office of the Under Secretary of Defense for Acquisition and Sustainment, 2021.

[217] N. G. Leveson and J. D. Reese, “Sample TCAS Intent Specification.” 1999. [Online]. Available: http://sunnyday.mit.edu/papers/tcas-intent.pdf

[218] P. Stanley and V. A. Barraquero, “STPA Evaluation of Potential Conflicts Between Large Commercial Air Traffic and Small Uncrewed Aircraft Systems in the Terminal Airspace,” presented at the STAMP Workshop, Massachusetts Institute of Technology, Cambridge MA, 2021.

[219] VWay Corporation, “VisualPro SA STPA,” 2023. https://eng.vway.co.kr/solutions/visualpro-stpa/

[220] Capella, “Arcadia Method - A Comprehensive Methodological and Tool-Supported Model-Based Engineering Guidance,” 2023. www.eclipse.org/capella/arcadia.html

[221] “NextGen Implementation Plan 2018-19.” Federal Aviation Administration, 2018. [Online]. Available: www.faa.gov/nextgen/media/NextGen_Implementation_Plan-2018-19.pdf

[222] FAA, “Time–Based Flow Management (TBFM).” www.faa.gov/air_traffic/publications/atpubs/foa_html/chap18_section_25.html (accessed Aug. 19, 2022).

[223] M. Ekal, K. Albee, B. Doerr, P. Roque, R. Ventura, and R. Linares, “MIT/IST/KTH ReSWARM,” presented at the NASA Spheres / Astrobee Working Group, 2022. [Online]. Available: www.nasa.gov/content/spheresastrobee-working-group

[224] M. G. Bualat, T. Smith, E. E. Smith, T. Fong, and D. Wheeler, “Astrobee: A New Tool for ISS Operations,” in *2018 SpaceOps Conference*, Marseille, France: American Institute of Aeronautics and Astronautics, May 2018. doi: 10.2514/6.2018-2517.

[225] C. H. Fleming and N. G. Leveson, “Improving Hazard Analysis and Certification of Integrated Modular Avionics,” *J. Aerosp. Inf. Syst.*, vol. 11, no. 6, pp. 397–411, Jun. 2014, doi: 10.2514/1.I010164.

[226] S. L. Estes, K. J. Burns, J. R. Helleberg, K. M. Long, M. E. Pollack, and J. L. Stein, “Digital Copilot: Cognitive Assistance for Pilots,” presented at the Association for the Advancement of Artificial Intelligence - Fall Series, 2016, pp. 141–144.

Appendix 1: Categorization Data for Set of Systems Analyzed

Table A1-2 shows categorization data of major controller interactions for systems sampled from the literature reviewed for this work. Each interaction is categorized according to the taxonomy of structure of interactions between controllers, and for the presence of the collaborative control dynamics described in Chapter 3. The legend in Table A1-1 explains the meaning of symbols in the data.

Table A1-1. Legend for Categorization Data

Header	**Symbol and Meaning**
Fielded	0: system is not fielded (still in various development stages) 1: system is fielded
Type of Controllers	HH: Human – Human Interaction HM: Human – Machine Interaction MM: Machine – Machine Interaction SS: Higher-level abstractions with combination of above
Hierarchical Structure	H: Hierarchal Control Interaction P: Peer Interaction HP: Mix of Hierarchal and Peer Interactions
Behavioral Intent	1: Primarily cooperative in intent 2: Primarily unknown and/or mixed-motive intent 3: Primarily adversarial in intent
Connectivity	G: Controllers globally connected L: Controllers locally connected N: Controllers not connected *Multiple: different modes of interaction can exist*
Information Exchange	A: Active (deliberate content-based messaging) P: Passive (observation only) N: No Information Exchange *Multiple: different modes or multiple simultaneous types of exchanges*
Roles & Responsibilities	P: Prescribed (control boundaries delineated to avoid overlap per mode) D: Dynamic (control boundaries overlap coordinated during execution) A: Ad-hoc (control boundaries not predefined, determined in execution)
Collaborative Control Dynamics	0: Dynamic not exhibited 1: Dynamic exhibited 0.5: Dynamic may or may not be exhibited in different system versions

Table A1-2. Categorization Data for Interactions in Systems Sampled

#	System (Interacting Entities)	Fielded	Structural Dimensions of Controller Interaction							Collaborative Control Dynamics								
			Type Controllers	Hierarchal Struct	Behavioral Intent	Connectivity	Info Exchange	Roles & Resp	Dev Origins	Cognitive Alignment	Lateral Coord	Mutual Close Loop	Shared Authority	Transfer Authority	Dynamic Authority	Dynamic Hierarchy	Dyn Membership	Dyn Connectivity
1	Cockpit Situation Display (Pilot – CSD) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
2	Enhanced Vision Sys (Pilot - EVS) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
3	Synthetic Vision Sys (Pilot - SVS) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
4	Combine Vision Sys (Pilot - CVS) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
5	External Vision Sys (Pilot - XVS) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
6	Heads Up/Mounted Display (Pilot - HUD/HMD) [23]	1	HM	H	1	L	A	P	2	0	0	0	0	0	0	0	0	0
7	Ground Prox Warning Sys (Pilot - GPWS) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
8	Terrain Awareness & Warning Sys (Pilot - TAWS) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
9	Cockpit Display of Traffic Info (Pilot–Display) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
10	Auto Dependent Surv Broadcast In (Pilot - ADS-B In) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	1
11	NextGen SURF-IA (Pilot – Display) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
12	RNAV & RNP Display Tools (Pilot – Tool) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
13	NextGen On-Demand NAS Info (Human – Tool) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
14	Overrun Protection Sys (ROPS) (Pilot – Tool) [23]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
15	Emergency Landing Planner (Pilot – ELP) [23]	0	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
16	Electronic Centralized A/C Monitor (ECAM) [23]	1	HM	H	1	G	A	P	1	0	0	0	0	0	0	0	0	0
17	Auto Ground Collision Avoidance Sys (Pilot – AGCAS) [23]	1	HM	H	1	G	A	P	2	0	0	0	1	1	0	0	0	0
18	NextGen Collaborative ATM (ATC-Dispatchers) [221]	1	HH	P	1	G	A	P	2	1	1	1	1	0	0	0	1	1
19	NextGen Collab ATM (ATC or Dispatchers - Pilot) [221]	0	HH	H	1	L	A	P	2	1	0	1	1	0	0	0	1	1
20	NextGen Collab ATM (Humans – Decision Tool) [221]	0	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	1	1
21	Time-Based Flow Management (ATC - ATC) [222]	1	HH	H	1	L	A	P	2	1	1	1	1	1	0	0	1	1
22	Time-Based Flow Management (ATC - Decision Tool) [222]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	1	1
23	Traffic Collision Avoid Sys (TCAS Aircraft - TCAS A/C) [138]	1	MM	HP	1	L	A	D	2	1	1	1	1	0	1	0	1	1
24	Traffic Collision Avoid Sys (TCAS – Flight Crew) [138]	1	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
25	Traffic Collision Avoid Sys (TCAS – ATC) [138]	1	MH	P	1	G	A	P	2	0	1	0	1	1	0	0	1	1
26	ACAS-X Active & Passive Coord (Peer-Peer Aircraft) [202]	0	MM	HP	1	L	A	D	2	1	1	1	1	0	1	0	1	1
27	ACAS-X Responsive Coord (Senior - Junior Aircraft) [202]	0	MM	HP	1	L	A	D	3	1	1	1	1	0	1	0	1	1
28	ACAS-X (ACAS Aircraft – Non-ACAS Aircraft) [202]	0	MM	P	12	L	P	D	3	0	1	0	1	0	0	0	1	1
29	ACAS-X (ACAS – Flight Crew, ACAS - Autopilot) [202]	0	HM	H	1	G	A	P	2	0	0	0	0	0	0	0	0	0
30	ACAS-X (ACAS – ATC) [202]	0	HM	P	1	G	A	P	2	0	1	0	1	1	0	0	1	1
31	UAS Detect And Avoid (Operator - DAA) [191]	0	HM	H	1	G	A	P	1,2	0	0	0	0	0	0	0	0	0
32	UAS Detect And Avoid (UAS - other Aircraft) [191]	0	SS	HP	12	L	A	D	3	1	1	1	1	0	1	0	1	1
33	UAS Detect And Avoid (ATC - Traffic) [191]	0	SS	H	1	L	AP	P	2	1	0	0	0	0	0	0	1	1
34	Patriot Friendly Fire (Aircraft - Patriot Battery) [191]	1	SS	P	1	L	AP	P	3	1	1	0	0	0	0	0	1	1
35	Patriot Friendly Fire (JFAC Air Cmd – JFLC Land Cmd) [191]	1	HH	P	1	L	AP	D	2	1	1	1	1	0	1	0	0	1
36	"Playbook" (Human - Automation) [127]	0	HM	H	1	GL	A	D	2	1	0	0	1	1	1	0	0	0
37	Common Autopilot or Auto-throttle (Pilot – Auto)	1	HM	H	1	G	AP	P	1	0	0	0	1	1	0	0	0	0
38	Full Authority Digital Engine Control (Pilot - FADEC)	1	HM	H	1	G	A	P	1	0	0	0	0	0	0	0	0	0
39	Future Multi-Rotor Flight Control System (Pilot - FCS)	0	HM	H	1	G	AP	P	1	0	0	0	0	0	0	0	0	0
40	Terrain Following Radar (Pilot - TFR)	1	HM	H	1	G	AP	P	1	0	0	0	1	1	0	0	0	0
41	Airliner Autoland (Pilot - Autoland)	1	HM	H	1	G	AP	P	1	0	0	0	1	1	0	0	0	0

#	System (Interacting Entities)	Fielded	Structural Dimensions of Controller Interaction							Collaborative Control Dynamics								
			Type Controllers	Hierarchal Struct	Behavioral Intent	Connectivity	Info Exchange	Roles & Resp	Dev Origins	Cognitive Alignment	Lateral Coord	Mutual Close Loop	Shared Authority	Transfer Authority	Dynamic Authority	Dynamic Hierarchy	Dyn Membership	Dyn Connectivity
42	Triple Modular Avionics on Adv Aircraft (Modules)	1	MM	P	1	G	A	P	1	0	0	0	1	0	0	0	0	0
43	Emergency Descent Mode (Pilot - EDM) [Garmin Ltd.]	1	HM	H	1	G	AP	P	2	0	0	0	1	1	0	0	0	1
44	Electronic Stability and Protection (Pilot - ESP) [Garmin Ltd.]	1	HM	H	1	G	AP	P	2	0	0	0	1	1	0	0	0	1
45	Emergency Autoland (Human – Autoland) [Garmin Ltd.]	1	HM	H	1	G	AP	P	2	0	0	0	1	1	0	0	0	1
46	Brake Control Unit (Flight Crew - BSCU) [50]	1	HM	H	1	G	AP	P	2	0	0	0	1	1	0	0	0	0
47	Common Single UAS Op (Operator - Autopilot)	1	HM	H	1	GL	AP	P	1	0	0	0	1	1	0	0	0	1
48	Common Single UAS Op (Operator – Operator Team)	1	HH	P	1	G	AP	D	2	1	1	1	1	1	1	1	1	1
49	Communications Denied UAS (Operator - Autopilot)	0	HM	H	1	LN	AN	P	1	0	0	0	1	1	0	0	0	1
50	Auto Track Cinematography UAS (Operator - UAS) [Skydio]	1	HM	H	1	L	AP	P	1,3,4	1	1	1	1	0	0	0	0	1
51	Multi-Munition (Munitions) [26]	0	MM	P	1	L	A	D	1	1	1	1	1	0	1	0	1	1
52	Multi-Munition (Operator - Munitions) [26]	0	HM	H	1	L	A	D	1		0	0	1	1	1	0	1	1
53	Common Distributed Multi-UAS (Operator - UAS) [24], [145]	0	HM	H	1	L	A	D	1	1	0	0	1	1	1	0	1	1
54	Common Distributed Multi-UAS (UASs) [24], [145]	0	MM	P	1	G	A	D	1	1	1	1	1	0	1	0	1	1
55	HIPC Multi-UAS Distributed Control (UASs) [83]	0	MM	P	1	L	A	D	1	1	1	1	1	0	1	0	1	1
56	USMA Service Academy Swarm Challenge (UASs) [148]	0	MM	P	1	G	A	D	1	1	1	1	1	1	1	0	1	1
57	Multi-UAS Hazard Analysis (UASs) [143]	0	MM	P	1	L	A	D	1	0.5	0.5	0.5	1	0	1	0	1	1
58	Leader-Subordinate Multi-UAS Control (UASs) [91]	0	MM	HP	1	L	A	D	1	0.5	0.5	0.5	1	1	1	0.5	1	1
59	Multi-UAS Light Shows (UASs)	1	MM	P	1	L	N	P	1	0	0	0	0	0	0	0	1	1
60	Multi-UAS Light Shows (Operator - UAS)	1	HM	H	1	L	A	P	1	0	0	0	0	0	0	0	1	1
61	UAS-UGV Teaming (UAS – UGV)	0	MM	P	1	G	A	P	1	1	1	0.5	1	0	0	0	0	1
62	Human Piloted Flight Formation	1	HH	HP	1	G	AP	P	1	1	1	1	1	1	0	0	1	1
63	Manned Unmanned Teaming (MUM-T) (UASs) [53]	0	MM	P	1	L	A	D	-	1	1	1	1	1	1	-	1	1
64	MUM-T (Team Lead - GCS) [53]	0	HH	P	1	GL	A	D	2	1	1	1	1	1	1	0.5	1	1
65	MUM-T (TL or GCS - UAS) [53]	0	HM	HP	1	L	A	P	1	1	1	1	1	0.5	1	0	1	1
66	MUM-T (Neural Net Control Sys – Runtime Assurance) [214]	0	MM	P	1	G	AP	P	2	1	1	1	1	1	0	0	0	0
67	MUM-T (Safety Pilot – Automated Formation Control) [214]	0	HM	H	1	G	AP	P	1	-	0	-	1	1	0	0	0	0
68	Tethered UAS (PIC - UAS) [192]	0	HM	HP	1	L	A	P	1	1	1	1	1	0	0	0	1	1
69	Tethered UAS (UASs) [192]	0	MM	P	1	G	A	D	1	1	1	1	1	0	1	0.5	1	1
70	Satellite Constellation (Satellites) [18]	1	MM	P	1	N	N	P	1	0	0	0	0	0	0	0	1	1
71	Satellite Constellation (Orbital - Payload Teams) [18]	1	HH	P	1	G	A	P	1	1	1	1	1	0	0	0	0	-
72	Satellite Constellation (Operator - Satellite) [18]	1	HM	H	1	L	A	P	1	0	0	0	0	0	0	0	1	1
73	Distributed Satellite Constellation (Satellites) [18]	0	MM	P	1	L	A	P	1	1	1	1	1	1	1	0	1	1
74	Distributed Formation Control Astrobee [223]	0	MM	P	1	G	AP	P	1	1	1	1	1	0	1	0	0	0
75	Astrobee ConOps(Human-Auto) [224]	1	HM	H	1	L	AP	P	1	0	0	0	0	0	0	0	0	1
76	Combined Product Transportation Sys (Operators) [195]	0	HH	P	1	G	A	-	1	1	1	1	1	0	-	-	0	-
77	Combined Product Transportation Sys (Operator - AGV) [195]	0	HM	H	1	L	A	P	1	0	0	0	0	0	0	0	0	0
78	Combined Product Transportation Sys (AGV - AGV) [195]	0	MM	P	1	L	P	P	1	1	1	0.5	1	0	0	0	0	1
79	Supply Vessel Dynamic Positioning (Operators) [193]	0	HH	P	1	L	A	D	2	1	1	1	1	1	1	1	1	1
80	Supply Vessel Dynamic Positioning (AGV-AGV) [193]	0	MM	P	1	L	AP	D	1	1	1	1	1	0	1	0	1	1
81	Supply Vessel Dynamic Positioning (Operator - AGV) [193]	0	HM	H	1	G	AP	P	1	0	0	0	1	1	0	0	0	0
82	Airliner Flight Crew (Pilots) [190]	1	HH	HP	1	G	AP	D	2	1	1	1	1	1	1	1	1	0
83	Integrated Modular Avionics (Flap - Thrust) [225]	0	MM	P	1	G	A	P	2	1	1	0	0	0	0	0	0	0

#	System (Interacting Entities)	Fielded	Structural Dimensions of Controller Interaction							Collaborative Control Dynamics								
			Type Controllers	Hierarchal Struct	Behavioral Intent	Connectivity	Info Exchange	Roles & Resp	Dev Origins	Cognitive Alignment	Lateral Coord	Mutual Close Loop	Shared Authority	Transfer Authority	Dynamic Authority	Dynamic Hierarchy	Dyn Membership	Dyn Connectivity
84	In-Trail Procedure (ITP) (Aircraft-Aircraft) [188]	1	SS	P	2	L	A	P	3	0	1	0	0	0	0	0	1	1
85	In-Trail Procedure (ITP) (ATC Controllers) [188]	1	HH	P	1	L	AP	D	2	1	1	1	1	1	1	0	0	1
86	NASA & JAXA Coordination (2 Organizations) [50]	1	HH	P	1	L	A	P	2	1	1	0	0	0	0	0	0	1
87	1994 Friendly Fire (AWACS controllers) [38]	1	HH	P	1	L	AP	D	1	1	1	1	1	1	1	0	0	1
88	Embedded Vehicle Software Subsys (Subsystems) [206]	1	MM	P	1	G	AP	P	2,3	0	0	0	1	0	0	0	0	0
89	Digital Copilot [Collaborative Checklist] (Pilot – Auto) [22]	0	HM	HP	1	G	AP	D	1	1	1	1	1	1	1	1	0	0
90	Digital Copilot [Cognitive Assistance] (Pilot – Auto) [226]	0	HM	H	1	G	A	P	2	1	0	0	0	0	0	0	0	0
91	Pilot Assistant [NATO] [41]	0	HM	H	1	G	A	D	1,2	1	0	0	0	0	0	0.5	0	0
92	Reduced Crew Ops Test (Pilot - Remote Pilot) [5]	0	HH	P	1	G	AP	D	2	1	1	1	0.5	0.5	0.5	-	1	1
93	Reduced Crew Ops Test (Human - Autonomy) [5]	0	HM	H	1	G	A	P	2	1	0	0	1	1	0	0	0	0
94	Co-Active Design Model (TBD - TBD) [2]	0	any	P	1	GLN	AP	PD	1	1	1	1	1	1	0.5	0	0	0.5
95	UAM Simplified Vehicle Ops (Human - Autonomy) [8], [10]	0	HM	HP	1	G	AP	D	1	1	1	0.5	1	1	1	1	0	0
96	UAM Remote Supervisory Ops (Humans) [8]	0	HH	P	1	-	A	D	2	1	1	-	1	1	1	-	1	1
97	UAM Remote Supervisory Ops (Human - Auto) [8]	0	HM	H	1	G	A	D	1	1	0	0	0	0	0	0	1	1
98	UAM Remote Supervisory Ops (Machines) [8]	0	MM	P	2	N	N	P	2	0	0	0	0	0	0	0	1	0
99	UAM Distributed ATM Concept (ATM-ATM) [204]	0	MM	P	1	L	A	D	1	1	1	1	1	1	1	0	1	1
100	UAM Distributed ATM Concept (ATM-UAM) [204]	0	MM	H	1	L	AP	D	1	1	0	0	0	0	0	0	1	1
101	Mosaic of Warfare (Overall Concept) [32]	0	HM	HP	1	L	A	D	1-4	1	1	1	1	1	1	1	1	1

Appendix 2: MUM-T Case Study Unsafe Combination of Control Actions (UCCAs)

Abstraction 2a – Combinations of Control Actions Provided by the Team

Abstraction 2a – Type 1-2: Team provides / does not provide control actions

Table A2-1. Abstraction 2a - Type 1-2 (Provide / Not Provide Command) – Abstracted

id	team	team	sid	same	context	relevant
1	¬fix	¬{fire,search}	1	0	*when there are mission tasks to execute [H3]*	2,3
2	fix	¬{fire,search}	2	0	*when target will compromise mission if fixed but not fired on [H3]*	2
3	¬fix	{fire,search}	3	0	*when target fired-on must be fixed [H3,H5]*	2
4	fix	{fire,search}	4	0	*when fixed target is different than target fired-on [H3,H5]*	2
5	¬fire	¬{fix,search}	0	1	*when there are mission tasks to execute [H3]*	2,3
6	fire	¬{fix,search}	0	3	*when target fired-on must be fixed [H3,H5]*	2
7	¬fire	{fix,search}	0	2	*when target will compromise mission if fixed but not fired on [H3]*	2
8	fire	{fix,search}	0	4	*when fixed target is different than target fired-on [H3,H5]*	2
9	¬search	¬{fix,fire}	0	1	*when there are mission tasks to execute [H3]*	2,3
10	search	¬{fix,fire}	5	0	*when engaging a known target is higher priority than searching [H3]*	1,2
11	¬search	{fix,fire}	6	0	*when searching for another target has higher priority [H3]*	1,2
12	search	{fix,fire}	0	0		

Table A2-2. Abstraction 2a - Type 1-2 (Provide / Not Provide Command) – Refined

Id	sid	ssid	TL	TL	UAS1	UAS1	UAS1	UASn	UASn	UASn	priority	context*
1	1	1	¬fire	¬search	¬fix	¬fire	¬search	¬fix	¬fire	¬search	1	when…
2	2	1	¬fire		fix	¬fire		¬fix	¬fire		1	when…
3	3	1	fire		¬fix	¬fire		¬fix	¬fire		1	when…
3	3	2	¬fire		¬fix	fire		¬fix	¬fire		1	when…
4	4	1	fire		fix	¬fire		¬fix	¬fire		1	when…
4	4	5	¬fire		fix	¬fire		¬fix	fire		1	when…
10	5	1	¬fire	search	¬fix	¬fire	¬search	¬fix	¬fire	¬search	1	when…
10	5	2	¬fire	¬search	¬fix	¬fire	search	¬fix	¬fire	¬search	1	when…
11	6	1	fire	¬search	fix	¬fire	¬search	¬fix	¬fire	¬search	1	when…
11	6	3	¬fire	¬search	fix	¬fire	¬search	¬fix	fire	¬search	1	when…
11	6	5	¬fire	¬search	fix	fire	¬search	¬fix	¬fire	¬search	1	when…
1	1	2	¬fire	¬search	fix	¬fire	¬search	fix	¬fire	¬search	2	when…
2	2	2	¬fire		fix	¬fire		fix	¬fire		2	when…
3	3	4	fire		fix	¬fire		fix	¬fire		2	when…
3	3	5	¬fire		fix	fire		fix	¬fire		2	when…
4	4	4	fire		fix	¬fire		fix	¬fire		2	when…
4	4	7	¬fire		fix	fire		fix	¬fire		2	when…
10	5	4	¬fire	search	fix	¬fire	¬search	fix	¬fire	¬search	2	when…
11	6	2	fire	¬search	fix	¬fire	¬search	fix	¬fire	¬search	2	when…
3	3	3	fire		¬fix	fire		¬fix	¬fire		3	when…
3	3	6	fire		fix	fire		fix	¬fire		3	when…
3	3	7	¬fire		¬fix	fire		¬fix	fire		3	when…
3	3	8	fire		¬fix	fire		¬fix	fire		3	when…
3	3	9	¬fire		fix	fire		fix	fire		3	when…
3	3	10	fire		fix	fire		fix	fire		3	when…
4	4	3	fire		fix	fire		¬fix	¬fire		3	when…
4	4	6	fire		fix	¬fire		¬fix	fire		3	when…
4	4	8	fire		fix	fire		fix	¬fire		3	when…
4	4	9	¬fire		fix	fire		¬fix	fire		3	when…
4	4	10	fire		fix	fire		¬fix	fire		3	when…
4	4	11	¬fire		fix	fire		fix	fire		3	when…
4	4	12	fire		fix	fire		fix	fire		3	when…
10	5	3	¬fire	search	¬fix	¬fire	search	¬fix	¬fire	¬search	3	when…
10	5	5	¬fire	¬search	¬fix	¬fire	search	¬fix	¬fire	search	3	when…
10	5	6	¬fire	search	¬fix	¬fire	search	¬fix	¬fire	search	3	when…
11	6	4	fire	¬search	fix	¬fire	¬search	¬fix	fire	¬search	3	when…
11	6	6	fire	¬search	fix	fire	¬search	¬fix	¬fire	¬search	3	when…
11	6	7	¬fire	¬search	fix	fire	¬search	¬fix	fire	¬search	3	when…
11	6	8	fire	¬search	fix	fire	¬search	¬fix	fire	¬search	3	when…
4	4	2	¬fire		fix	fire		¬fix	¬fire		4	when…

* Context removed for legibility of the table, but traceable to the abstracted Table

Abstraction 2a – Type 3-4: Control actions start / end too early / too late relative to one another

Table A2-3. Abstraction 2a - Type 3-4 (Start / End Command too Early / Late) – Abstracted

id	Team ...	then ...	sid	same	context	relevant
13	S(fix)	S(fire,search)	0	0		
14	S(fix)	E(fire,search)	0	0		
15	E(fix)	S(fire,search)	1	0	*when target fired on must be fixed [H3,H5]*	2
16	E(fix)	E(fire,search)	0	0		
17	S(fire)	S(fix,search)	2	0	*when target fired on must be fixed [H3,H5]*	1
18	S(fire)	E(fix,search)	0	0		
19	E(fire)	S(fix,search)	0	0		
20	E(fire)	E(fix,search)	0	0		
21	S(search)	S(fix,fire)	0	0		
22	S(search)	E(fix,fire)	0	0		
23	E(search)	S(fix,fire)	0	0		
24	E(search)	E(fix,fire)	0	0		
25	S(fire,search)	S(fix)	0	2	*when target fired on must be fixed [H3,H5]*	2
26	S(fire,search)	E(fix)	0	0		
27	E(fire,search)	S(fix)	0	0		
28	E(fire,search)	E(fix)	0	0		
29	S(fix,search)	S(fire)	0	0		
30	S(fix,search)	E(fire)	0	0		
31	E(fix,search)	S(fire)	0	1	*when target fired on must be fixed [H3,H5]*	1
32	E(fix,search)	E(fire)	0	0		
33	S(fix,fire)	S(search)	0	0		
34	S(fix,fire)	E(search)	0	0		
35	E(fix,fire)	S(search)	0	0		
36	E(fix,fire)	E(search)	0	0		

Table A2-4. Abstraction 2a - Type 3-4 (Start / End Command too Early / Late) – Refined

id	sid	ssid	TL	UAS1	UAS1	UASn	priority	Context*
15	1	1	F S(fire)	E(fix)			1	when...
15	1	3		E(fix)		F S(fire)	1	when...
17	2	1	S(fire)	F S(fix)			1	when...
17	2	3		F S(fix)		S(fire)	1	when...
15	1	2		E(fix)	F S(fire)		4	when...
17	2	2		F S(fix)	S(fire)		4	when...

* Context removed for legibility of the table, but traceable to the abstracted Table

Abstraction 2b – Combinations of Controllers Issuing Common Control Action

Abstraction 2b – Type 1-2: Controllers provide / don't provide control actions

Table A2-5. Abstraction 2b - Type 1-2 (Provide / Not Provide Command) – Abstracted

id	Ci	Cj(s)	sid	same	context
37	¬fix	¬fix	1	0	*when there is a priority target to engage and a teammate able to fire [H3]*
38	fix	¬fix	2	0	*when tasked entity is not capable and another is [H3]*
39	fix	fix	3	0	*when that creates mutual interference [H1,H3]*
40	¬fire	¬fire	4	0	*when there is a priority target to engage and a teammate able to fix [H3]*
41	fire	¬fire	5	0	*when tasked entity is not capable and another is [H3]*
42	fire	fire	6	0	*when that creates excessive effects [H3,H5]*
43	¬search	¬search	7	0	*when no targets have been found [H3]*
44	search	¬search	8	0	*when tasked entity is the only one capable for higher priority task and teammate can search [H3]*
45	search	search	0	0	

Table A2-6. Abstraction 2b - Type 1-2 (Provide / Not Provide Command) – Refined

id	sid	ssid	TL	TL	UAS1	UAS1	UAS1	UASn	UASn	UASn	pri	context*
37	1	1			¬fix			¬fix			1	when…
38	2	1			fix			¬fix			1	when…
39	3	1			fix			fix			1	when…
40	4	1	¬fire			¬fire			¬fire		1	when…
41	5	1	fire			¬fire			¬fire		1	when…
41	5	2	¬fire			fire			¬fire		1	when…
42	6	1	fire			fire			¬fire		1	when…
42	6	2	¬fire			fire			fire		1	when…
43	7	1		¬search			¬search			¬search	1	when…
44	8	1		search			¬search			¬search	1	when…
44	8	2		¬search			search			¬search	1	when…
42	6	3	fire			fire			fire		3	when…

* Context removed for legibility of the table, but traceable to the Team-Level Table

Abstraction 2b – Type 3-4: Controllers start / end providing control actions too early / too late relative to one another

Table A2-7. Abstraction 2b - Type 3-4 (Start / End Command too Early / Late) – Abstracted

id	Ci ...	then Cj ...	sid	same	context
46	S(fix)	S(fix)	0	0	
47	S(fix)	E(fix)	1	0	*when that creates mutual interference [H1,H3]*
48	E(fix)	S(fix)	2	0	*when that creates a large gap in a fix handoff [H3]*
49	E(fix)	E(fix)	0	0	
50	S(fire)	S(fire)	0	0	
51	S(fire)	E(fire)	0	0	*N/A: Fire Discrete Command*
52	E(fire)	S(fire)	0	0	*N/A: Fire Discrete Command*
53	E(fire)	E(fire)	0	0	*N/A: Fire Discrete Command*
54	S(search)	S(search)	0	0	
55	S(search)	E(search)	0	0	
56	E(search)	S(search)	3	0	*when that creates an excessive gap in the search [H3]*
57	E(search)	E(search)	0	0	

Table A2-8. Abstraction 2b - Type 3-4 (Start / End Command too Early / Late) – Refined

id	sid	ssid	TL	UAS1	UAS1	UASn	UASn	pri	context*
47	1	1		F E(fix)		S(fix)		1	when…
48	2	1		F S(fix)		E(fix)		1	when…
56	3	1	E(search)		F S(search)			1	when…
56	3	2	F S(search)		E(search)			1	when…
56	3	3			F S(search)		E(search)	1	when…

* Context removed for legibility of the table, but traceable to the Team-Level Table

Appendix 3: MUM-T Case Study - Unsafe Causal Scenarios

The following analysis presents causal scenarios for the Manned-Unmanned Teaming case study that are developed (1) using a process of iterative refinement and (2) in a non-linear sequence.

Iterative refinement is accomplished using the process defined in Chapter 4.3. It first develops *top-level scenarios* to reason about the different potential control actions internal to the team that relates to the output in the UCCA (Step 1). Next, each scenario is refined by systematically exploring causal factors associated with these internal feedback control loops (Step 2) and those related to collaborative control dynamics (Step 3). Finally, other factors are identified that relate to unsafe control paths to the process, unsafe feedback from the process, or unsafe process behaviors (Step 4). The refinement template defined in Figure 4.21 helps add details to these scenarios as necessary.

Scenarios are also developed in a non-linear sequence. This means that scenarios are initially developed to address the UCCA listed above it. However, over the course of the analysis, as more UCCAs are analyzed and found to have similar scenarios, the original scenarios are modified to integrate some of those variations. The modified, and more inclusive scenarios are then also traced to those UCCAs. The intent is to consolidate multiple similar scenarios together, make the analysis more readable, and enhance the development of design requirements.

As an example, an excerpt of Scenario S-37.1.1 for UCCA 37.1 "C_i does not fix and no other C_j fixes when …" (see Example 1 above) is originally written as:

> **Initial Version - S-37.1.1 *Unsafe Control Input:*** TL misinterprets direction from higher authorities that the team should not fix any targets.

Several UCCAs subsequently analyzed have similar control combinations, such as UCCA 40.4 "C_i does not fire and no other C_j fires when …" and UCCA 43.7 "C_i does not search and no other C_j searches when …". To avoid repetition, the excerpt of S-37.1.1 above is modified to the form below, which covers the later UCCAs. Then, the matching scenario headers under S-40.4.1 and S-48.7.1 are traced to S-37.1.1. The following excerpt of S-37.1.1, which is not traced to any other scenario, is then labeled as a *unique scenario* in the data analysis performed in Section 5.5.

> **Modified Version - S-37.1.1 *Unsafe Control Input:*** TL misinterprets direction from higher authorities that the team should not <fix, fire on, search for> any targets.
>
> **S-40.4.1 *Unsafe Control Input:*** see S-37.1.1.
>
> **S-48.7.1 *Unsafe Control Input:*** see S-37.1.1.

Abstraction 2b – Combinations of Controllers Providing Shared Control Actions

Abstraction 2b – Type 1-2: Controllers provide / don't provide control actions

The following are the causal scenarios associated with all Abstraction 2b – Type 1-2 UCCA. Each high-level UCCA is analyzed together with its refined UCCAs. Scenario identification for each UCCA starts with the following six top-level scenarios for all Type 1-2 UCCAs. These are intended to provide focus and coverage over different possible control actions internal to the team that could lead to the unsafe collective output.

1. Direction Not Provided (Unsafe): TL does not direct the UAS as necessary for the team to execute safe collective control of the shared process.
2. Direction Provided (Unsafe): TL directs the UAS in a way that leads to unsafe collective control. This includes:
    a. TL directs multiple UAS to provide commands that conflict with one another.
    b. TL directs the UAS to provide an incorrect command.
    c. TL directs an incorrect UAS to provide a command.
3. Direction Provided (Safe) but Not Executed Properly (Unsafe): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. This includes:
    a. UAS do not execute some of the tasks provided.
    b. UAS execute incorrectly some of the tasks provided.
    c. Incorrect UAS execute some of the tasks provided.
4. Direction Not Provided (Safe) but Executed (Unsafe): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control.
5. Controller Actions to Process and Directions it Provides (Unsafe): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. This includes:
    a. TL does not provide a control action that is necessary in combination with actions it directed to the UAS.
    b. TL provides a control action incorrectly or in a way that conflicts with actions it directed to the UAS.
9. Other factors beyond those explored in interactions between the controllers on the team.

UCCA 37.1 (abstracted UCCA): Controller Ci does not provide fix and no other Cj provides fix when there is a priority target to engage and a teammate able to fire [H3] [DA]

UCCA 37.1.1 (refined UCCA): UAS1 does not provide fix and UAS2 does not provide fix

Figure A3-1Figure 5-4 shows the four relevant internal control combinations that can lead to the collective output of the UCCA. It enumerates whether or not the Team Lead (TL) tasks either UAS_1 or UAS_n to fix. The figure is only provided here to demonstrate how the internal control combinations are captured into the top-level scenarios, as listed below each case in the figure.

However, the analyst could go straight into listing the top-level scenarios. Because UAS_1 and UAS_n are interchangeable, case 1c in the figure is a duplication of 1b.

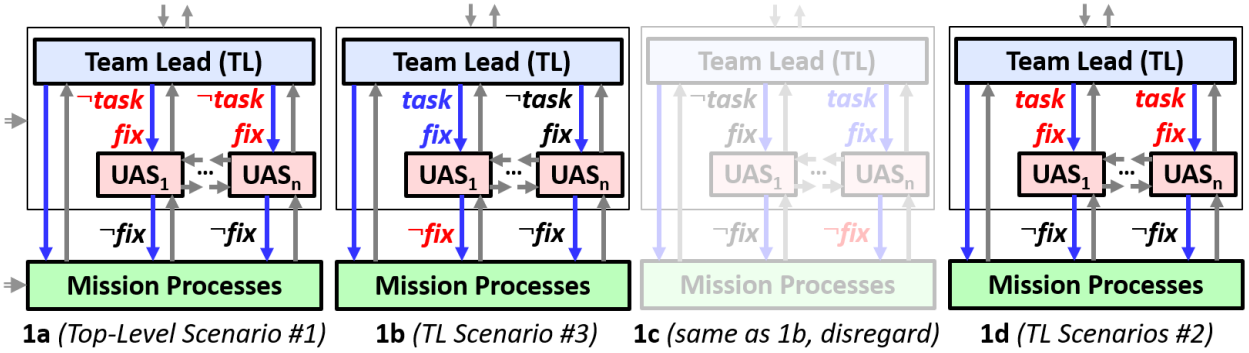


Figure A3-1. Internal control combinations that result in no controller providing a fix

S-37.1.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, the TL does not task any UAS to fix.[7] [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL misinterprets direction from higher authorities that the team should not <fix on, fire on, search for> any targets. ***Refinement***:
  - TL previously received a command not to provide a <fix, fire, search>. However, that command is now outdated, but the TL does not receive the updates.
  - TL receives a correct command to <fix on, fire on, search for > targets as necessary, but misinterprets it due to ambiguous communications.
  - …
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has one or more of the following inadequate process model variables: TL does not believe (1) there is a target to <engage, search for>, (2) there is a teammate able to fire (to couple with fix), (3) that a UAS has already been tasked to <fix, search>, or (4) there is a teammate available to task the <fix, search>. ***Refinement***:
  - *Incorrect feedback.* The interface provides stale feedback of the team state, which shows one of the UAS as tasked to <fix, fire, search> from a now outdated tasking.
  - *Incorrectly Interpreted Feedback.* TL misinterprets sensor data about the target, due to heavy workload, and believes there is no target to <engage, search for>.
  - *Feedback not sent.* UAS are temporarily disconnected, so TL is not aware there is a UAS able to <fix, fire, search>.
  - …

  (Step 2: Internal Control) Inadequate Control Algorithm: TL has accurate information about the target and the team capabilities, but still chooses not to task a UAS to issue the fix. ***Refinement***:
  - TL is currently busy and prioritizes other operating tasks, but intends to task a UAS soon. However, s/he later forgets to do this due to high workload.

[7] Baseline (B) UCA 1: TL does not provide fix command to UAS when targets/enemies are within range

- TL misunderstands how the UAS operate. S/he believes that the UAS tasked to fire will by default provide its own fix.

- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task a UAS, but is unable to do so. ***Refinement***:
  - The communication channel to do so is currently inadequate. This could be due to RF jamming, communications fading, misconfigured encryption settings, or other reasons.
  - The interface has a complicated or confusing workflow to specify a task. This is aggravated in a high workload environments.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*****:** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*****:** *Not applicable. Top-level scenario focuses on a control decision from the TL only.* (See Inadequate Process Model / Control Algorithm)
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
  - TL intends to task a UAS, however, the set of UAS participating on the team fluctuates too much at the moment for the TL to have sufficient confidence that the assigned UAS will still be in the team to carry out the task. ***Refinement***:
    - The Ground Station is swapping out UAS that need to be refueled with new ones.
  - TL incorrectly anticipates that a new UAS will be added to the team that is better suited to execute the <fix, fire, search> task. S/he purposefully holds off on providing the task until that UAS is online. However, that UAS never arrives and the task goes unfulfilled. ***Refinement***:
    - The new UAS has a malfunction before it can join the team and must return to base. However, the TL is not notified.
  - TL tasks a UAS. However, that UAS is momentarily removed from the team. When it rejoins, TL incorrectly assumes that the UAS will resume its previous tasking, and takes no further action (e.g., retask the same UAS, task another UAS).
  - TL intends to task a UAS, however, the authority over that UAS is transferred to a different Team Lead, and it can no longer be tasked as part of the original team.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
  - TL is uncertain if tasking a UAS to <fix, fire, search> will result in the UAS traveling to a location where it will be disconnected from the team. As such, TL chooses to not provide the tasking.
  - TL tasks a UAS to <fix, fire, search>, but due to the dynamic topology, the tasking is not adequately routed to the intended UAS(s). This could occur because the message is degraded over too many hops, or because the changing topology does not support information routing requirements (e.g., path, timing, …).[8]

[8] (B) Scenario: Command corrupted in transmission by fault injection or SQL injection

S-37.1.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, (1) the TL tasks multiple UAS to fix and therefore none execute it, or (2) the TL tasks a UAS to fix that cannot fix. [DA]

Note: *because (2) is emphasized in the context of UCCA 38.2, this scenario focuses on item (1).*

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL interprets a command from higher authority as direction to task multiple UAS to <fix, fire>.
    - TL had already tasked a UAS to <fix, fire>, but it then receives additional direction from higher authorities that it interprets as direction to have a specific controller provide the <fix, fire> command, which is different than the one already tasked.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has one or more of the following inadequate process model variables: (1) a UAS has not been tasked to <fix, fire> when it actually has, (2) multiple UAS will not cause hazardous effects if they <fix, fire> simultaneously, or (3) what the set of active controllers is at any given time. ***Refinement:***
    - *Delayed feedback.* TL tasks UAS1 to <fix, fire>, but feedback from UAS1 acknowledging the tasking is delayed back to the TL (e.g., degraded communication channel, hidden in layers of the interface, …). As a result, the TL incorrectly believes UAS1 was not tasked, and tasks UAS2.[9]
    - *Feedback Not Available.* The UAS detect that multiple of them are tasked to <fix, fire>, which will create hazardous effects <interference, excessive damage>. However, the system is not designed to alert the TL of this problem so that s/he can take corrective action.
    - *Feedback Not Received.* UAS1 and UAS2 are part of the team. Feedback from UAS2 is not received by the TL, so TL believes UAS2 is not on the team. As such, TL provides the <fix, fire> command without specifying the intended UAS because s/he believes UAS1 is the only one that can be tasked. However, UAS2 does receive the task and executes it in addition to UAS1.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the target and the team, but still chooses to task multiple UAS to issue the fix. ***Refinement:***
    - TL misunderstands how the UAS operate and believes they will coordinate among themselves to determine which one takes on the task (similar to how the allocation occurs for the search command).[10]
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task a single UAS but inadvertently tasks multiple UAS or the wrong UAS. ***Refinement:***
    - The tasking interface is inadequate given the workload and operating environment.[11]

[9] (B) Scenario: The FMS is experiencing delayed processing and is unable to update target selections
[10] (B) Scenario: TL incorrectly believes that another UAV can fix on the target if he selects for the UAV to perform the command (incorrect assumption about the FMS control algorithm)
[11] (B) Scenario: TL chooses a different UAV because he is confused by the user interface

    - TL had originally tasked UAS1 to <fix, fire> but changes the plan and instead tasks UAS2. However, the communication channel to UAS1 is degraded and it does not receive the task cancellation. Both UAS are now tasked.[12]
    - The identification parameters of UAS2 are inadvertently reset. UAS2 now consumes all information intended by TL for UAS1.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*:** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*:** *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - TL tasks UAS1 to <fix, fire>. UAS1 is temporarily taken offline from the team (e.g., it is temporarily redirected by the Ground Station). So, TL tasks UAS2 to <fix, fire>. But then UAS1 rejoins the team and resumes its task.
    - TL intentionally tasks multiple UAS to compensate for fluctuations in the participation of the UAS.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
    - TL tasked UAS1 to <fix, fire>. UAS1 travels out to execute its task, and in the process loses connectivity with the team. The TL creatively implements a solution to regain connectivity by tasking UAS2 to the same task. S/he anticipates that UAS1 and UAS2 will either interfere with each other (if fix only) or they will recognize via coordination that they have the same task. This should prompt them to drop their tasks and reconnect with the team.

S-37.1.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks a single capable UAS to fix, but the UAS does not fix.[13] [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:***
    - The UAS tasked to <fix, fire, search> is overridden by another controller (e.g., the Ground Station, a different Team Lead, or a cyber attacker) to perform a different action.
- ***(Step 2: Internal Control) Inadequate Process Model:*** The tasked controller has the following inadequate process model variable: (1) it does not know how to provide the <fix, fire, search> command, (2) it incorrectly believes the <fix, fire, search> command has already been provided.
    - *Inadequate Feedback:* The controller <TL, UAS> tasked to provide the <fix, fire, search> has inadequate sensor feedback of the <target, search area> to execute the fix.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.

[12] (B) Scenario: There is a delay in another command from the Team Lead to delete the command
[13] (B) UCA 29: UAS does not implement a task from the TL (H3)

- ***(Step 2: Internal Control) Unsafe Control Path:*** the controller <TL, UAS> tasked to <fix, fire, search> intends to provide the assigned command. However, it does not deliver the command to the controlled process. ***Refinement:***
    - Its <targeting, weapon system, search sensor> equipment malfunctions and is unable to do so.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** In this system, if the controller that provides the fix is different from the controller that fires, those two controllers work collaboratively to fix and fire on the target. These two control loops are coupled as represented by Figure A3-2. ***Refinement:***

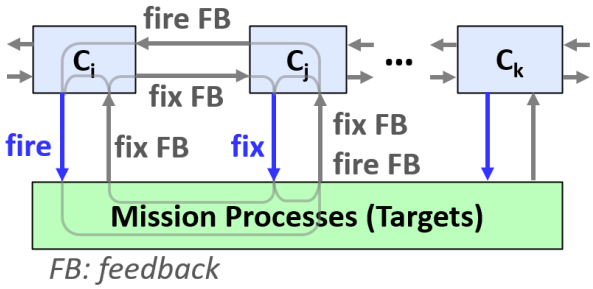


Figure A3-2. Control structure for controllers collaborating by fixing and firing on a target

- ***Feedback about Controlled Process from Collaborators*:** The feedback the UAS tasked to fix receives from collaborators leads it to believe it does not need to provide the <fix, fire> command. ***Refinement:***
    - The UAS tasked to fix (e.g., UAS1), prior to starting the fix, receives incorrect feedback from a teammate that the current fix is adequate to fire. Based on this feedback UAS1 does not provide the fix. ***Refinement***:
        - The teammate broadcasts feedback for a different fix, of a different target, provided by a different teammate.
        - (Human-Machine) UAS1 provides the fix for the TL (human) to fire. The TL inadvertently labels feedback for UAS1's fix as adequate. This may occur because the TL is under heavy workload and makes a mistake. It could also occur because there is a human-machine semantic mismatch in the feedback.
        - UAS1 provides inadequate feedback regarding the fire command to the teammate responsible for firing (e.g., informs the target is destroyed). Reasons for this inadequate feedback can be further refined. The teammate decides it will no longer fire and therefore, it provides feedback that the current fix from UAV1 (i.e., no fix) is adequate. As such, UAV1 continues to not provide a fix.
    - The UAS tasked to fix (e.g., UAS1), receives feedback from a teammate other than the one assigned to fire that the fix is inadequate. As a result, UAS1 provides a fix in response to the feedback from that other teammate, instead of providing a fix adequate for the teammate assigned to fire.

- ***Feedback about Collaborator Control Actions from Controlled Process*:** The feedback the UAS tasked to fix receives from the process leads it to believe it does not need to provide the fix command. ***Refinement:***
  - The UAS tasked to fix a target (e.g., UAS1) temporarily receives fix energy off the target provided by another UAS (e.g., UAS2). This leads UAS1 to drop the task in the belief that UAS2 is executing it. However, this may have been a stray command from UAS2 (e.g., in the process of fixing a different target, temporary system malfunction, …). As a result, no UAS provides the fix command as tasked by the TL.

  ***Note**: The UCCA analyzed does not explicitly show the fire command. If this control action coupling was not considered in this step, it would be explicitly covered in UCCA 3.3, in which the Team provides the fire command and does not provide the fix command.*

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. Figure A3-3 illustrates a possible misalignment of hypothetical model elements between MUM-T controllers to conceptualize this dynamic for analysis.

HUMAN - TL Mental Model (example)			
Controlled Process	Fix+Fire: Tgt A, Time B, Procedure C		
Collaborators			
Set of Controllers	TL	UAV1	UAV2
States	$\{xyz...\}_{\sim t}$	$\{xyz...\}_{\sim t-1}$	$\{xyz...\}_{\sim t}$
Responsibilities	Task UAVs Fire	Fix	-none-
Connectivity	N/A	Direct	Indirect
Info Req (to)	N/A	Task Coord	Tasking
Info Req (from)	N/A	Task Coord	Status
Confidence	High	High	Low
Environment	…		
Other Controllers	…		

MACHINE - UAV1 Process Model (example)			
Controlled Process	Fix+Fire: Tgt A, Time B, Algorithm D		
Collaborators			
Members	TL	UAV1	UAV2
States	$\{xyz...\}_{t}$	$\{xyz...\}_{t}$	$\{xyz...\}_{t}$
Responsibilities	Task UAVs -none-	Fix	Fix
Connectivity	Direct	N/A	Direct
Info sent to	Status	N/A	Task Coord
Info received from	Tasking	N/A	Task Coord
Confidence	High	Low	High
Environment	…		
Other Controllers	…		

Figure A3-3. Misalignment of Hypothetical Models between MUM-T Controllers

- ***Construction***: The process models and/or control algorithms are not adequately or consistently built across the team to support collaborative control. ***Refinement:***
  - (Human-Machine) The control algorithms on the UAS are not compatible with how the TL specifies the <fix, fire> task. This prevents the controller from accepting the task or being able to collaborate with another controller in a coupled <fire, fix> task. This could occur due to configuration management problems with the UAS or TL interface software, variation in how the TL was trained to work with the UAS, and conflicting past experience the TL has in working with human wingmen instead.[14]

[14] (B) Scenario: The AuC cannot interpret the command due to a recent firmware update

        - The control algorithms on the different UAS are not compatible with one another. As a result, the UAS lack the ability to coordinate in coupled fix & fire tasks. This could be due to a configuration management issue.
    - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. Refined:
        - The different UAS receive different versions of the <fix, fire> task specification by the TL. This could occur because the TL periodically updates how the task is specified given slight changes in her/his model of the mission. The different UAS may receive the task specification across different replan cycles. As such, the controllers do not have sufficient common knowledge of the task to coordinate the coupled fix & fire tasks.
        - The different UAS have different beliefs about how many and which UAS are participating on the team. This may occur due to dynamic membership, with UAS cycling on and offline, and dynamic connectivity that prevents the timely distribution of this information. This could result in the UAS assigned to fix (e.g., UAS1) not believing that the UAS assigned to fire (e.g., UAS2) is currently participating in the team. As such, UAS1 does not believe conditions are satisfied for it to provide a fix.
        - UAS have different beliefs about the current roles and responsibilities of other controllers on the team.
            - For example, one UAS believes that UAS2 has taken on the <fix, fire> task based on stale information, and disseminates this belief to the rest of the team. As a result, UAS1 loses confidence in its assignment and drops the <fix, fire> task.[15]
            - As another example, the controller assigned to <fix, fire> is not informed about which teammate is responsible for the coupled <fire, fix> task. As a result, it does not know whom to coordinate the coupled task with.
    - ***Model Updates***: elements of the process models are not adequately or consistently updated across the team. ***Refinement***:
        - *Vertical Coordination (Control):*
            - The TL controls the coordination details between UAS1 (tasked to fix) and UAS2 (tasked to fire). However, the information the TL provides is inconsistent between UAS1 and UAS2, or it conflicts with their ability to execute the coupled task.
            - The TL overrides one of the UAS with a command inconsistent with the laterally coordinated task execution. For example, UAS1 (tasked to fix) and UAS2 (tasked to fire) have laterally coordinated the details of the coupled fix and fire tasks. The TL then accelerates the UAS2 timeline to make it available for other tasks, but s/he is not aware that this now conflicts with the coordinated plan.

[15] (B) Scenario: The AuC has an incorrect process model that the target is already fixed on.

- TL keeps changing the plan and delegates different UAS to <fix, fire, search> before they have the opportunity to carry out the tasks.

- *Lateral Coordination (Communication):*
  - Communication channels between the UAS assigned to fix and the controller assigned to fire are not adequate to coordinate the coupled tasks. Too many coordination messages needed for the UAS to reach consensus on when or how to provide the commands are dropped. Communication channels may be degraded due to jamming, fading, and equipment damage.
  - (Human-Machine) Lateral coordination between UAS1 (tasked to fix) and TL (self-tasked to fire) is hindered by human-machine asymmetry in information semantics. Both controllers interpret shared information differently. The TL encodes subtleties and ambiguities, which are common for humans to handle, but difficult for machines to process precisely. The UAS is unable to describe task parameters that exceed its programmed bounds, but which may be necessary to overcome unforeseen issues. This hinders the execution of the coupled fix-fire commands.
  - (Human-Machine) Lateral coordination between UAS1 (tasked to fix) and TL (self-tasked to fire) is hindered by human-machine asymmetry in information timing. For example, the machine provides excessive information requests, which interrupt the flow of the TL in the execution of the coupled fix-fire commands.
  - The information controllers use to coordinate the coupled fix and fire tasks is inconsistent. For example, UAS1 (tasked to fix) receives a state estimate from UAS2 (tasked to fire) that becomes slightly outdated due to small communications and processing delays. For planning, it then compares this now slightly outdated UAS2 state estimate with its own current UAS1 state estimate and incorrectly concludes that UAS2 is lagging too much to provide the fix command. The same effect also occurs when UAS2 plans to fire.[16]
  - UAS share parameters about different targets without specifying which target they are describing. Each teammate then incorrectly associates that information with the target they are focused on. This creates disturbances in the process model, which may contribute to a UAS not executing its task.
  - UAS2 is temporarily disconnected and its model of the target diverges from that of the team. UAS2 then reconnects and contributes its divergent variables. That disturbs team consensus

[16] (B) Scenario: Autonomous Controller has incorrect feedback about the UAV's attitude, altitude, velocity. The FMS is sending data delayed so the AuC is calculating states behind where the UAV is actually located or its current speed.

regarding how to engage the target, changes the process model of UAS1 assigned a task, and contributes to not doing it.

- *Lateral Coordination (Observation)*: The <UAS, TL> controller selected to execute the <fix, fire, search> task observes another controller maneuvering in a way that is consistent with executing that task and incorrectly believes it will provide it. The maneuvering controller is not aware of this misinterpretation. As a result, the assigned controller does not provide the <fix, fire, search> task.[17]
- *Prediction*: UAS_1 (tasked to fix) has a model of the coupled fix & fire task that expects to receive certain coordination messages and observations by certain milestones. If the other controller <TL, UAS2> (tasked to fire) is delayed or takes an unexpected trajectory to fire, UAS1 may lose confidence in its ability to fulfill the task, and therefore incorrectly decide not to provide the fix task either.[18]

o ***Decision-Making:*** The process controllers use to decide what control and communications actions they provide is inadequate or inconsistent across the team. ***Refinement:***

- Despite adequate communication channels, the distributed decision-making process is too slow to keep up with the dynamic state of the shared-controlled process, and does not converge fast enough on a solution.
  - Two controllers attempt to coordinate the coupling of fix and fire commands. However, at the conclusion of every iteration of distributed planning, the state of the system has changed enough that the plan is no longer relevant. As a result, the team is unable to reach a valid plan.
- Despite adequate information sharing between the UAS, the control algorithm "churns". It replans or reoptimizes too frequently, overrides previous solutions before they can be executed, and ultimately causes the <fix, fire> tasks to not be performed.[19]
  - For example, UAS1 (tasked to fix) is at location that influences where <TL, UAS2> (tasked to fire) should be for the coupled task. However, by the time <TL, UAS2> reaches its new location, UAS1 has repositioned, which prompts a replan. The process repeats itself and the task is not executed.
- Controllers on the team share a relatively consistent set of common information. However, the automated controllers employ a non-deterministic decision-making algorithm (e.g., Machine Learning based). This leads to unpredictable or inconsistent control and information

[17] (B) Scenario: TL notices UAV(s) begin following target so he believes they are already tracking target
[18] (B) Scenario: machine learning algorithm has a false positive because the command occurred at an unusual time in the mission
[19] (B) Scenario: The AuC uses a machine learning algorithm that dynamically changes which UAV should implement the task too often such that it cannot select a UAV to perform the task

outputs. As a result, consensus on how to effectively execute a coupled fix and fire task may not be achievable.[20]

- Controllers on the team have a relatively consistent set of common current information. However, the controller (e.g., UAS1) assigned to the <fix, fire> task has low confidence in the behavior of its partner (e.g., UAS2) in the joint activity assigned to <fire, fix>. This low confidence may be due to many of the factors listed above. As a result, UAS1 precautionarily chooses not to <fix, fire>.[21]
- Controllers have a common set of planning information and are configured with common, deterministic, decision-making algorithms. However, they differ in their contextual factors, and as a result, use the decision-making algorithm differently.
    - For example, UAS1 may be dealing with an internal system malfunction, or maybe be targeted by an enemy, and prioritizes resolving those issues over collaborating with <TL, UAS2>.
    - (Human-Machine) As another example, the human TL recognizes a contextual nuance in the joint engagement process (coupled fix & fire task) that requires adapting the typical procedure. However, such changes cannot be effectively conveyed to an automated controller.

- ***Capacity:*** The capacity of one of the controllers is inadequate to enable the effective alignment of team cognition. Refined:
    - One of the controllers, TL or UAS, has a runaway internal process that is using up processing resources (e.g., system failure, denial of service). As a result, the controller is unable to keep up with the coordination demands of the rest of the team, which prevents the whole team from being able to reach consensus or achieve execution goals.[22]

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - The UAS tasked to <fix, fire, search> is not operating in a control mode that enables it to take on mission tasks as part of the team (e.g., "Team Mode").[23]
        - For example, it is operated in a mode that only accepts basic waypoints and not higher-level tasks.
        - As another example, the identifier of the UAS may be different from the one the TL and other members of the team are tracking. As a result, the TL does not consider the UAS to be part of the team.

---

[20] (B) Scenario: AuC machine learning algorithm determines that the [payload] is insufficient to complete the task and does not power it on as a result

[21] (B) Scenario: TL does not provide fire command because does not trust UAV(s) to accurately hit targets.

[22] (B) Scenario: software algorithm is busy processing requests from the Ground Station; MFC processor is slowed due to high temperatures and consistent operation.

[23] (B) Scenario: AuC cannot accept command without a prerequisite on the aircraft (e.g., master arm on).

- The UAS tasked to <fix, fire, search> is removed from the team. For example, it is overridden by another controller, such as a Ground Station, to return to base.[24]
- The loss of a teammate makes the UAS tasked to <fix, fire> drop its task. For example, at the time the UAS was tasked to fix (e.g., UAS1), another controller was tasked to collaborate with it by firing (e.g., UAS2). UAS2 is taken offline (by another controller, lost in mission, …) and its fire task is reassigned. However, the retask is not made known to UAS1, which therefore does not know whom to coordinate with to couple its fix task.
- The addition of a teammate makes the UAS tasked to <fix, fire> drop its task (e.g., UAS1). For example, since the time UAS1 was tasked, a new UAS is added to the team, which is better positioned for the task. UAS1 has logic to anticipate a retasking from the TL and drops the task. However, the task is never reassigned.

- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
  - The process of providing the <fix, fire> command leads UAS1 to temporarily disconnect from the team (e.g., physically traveling to the location to <fix, fire> causes it to disconnect). Because connectivity is required to coordinate the coupled fix-fire tasks with another controller, UAS1 drops its task and does not reclaim it when connectivity is regained.
  - The UAS tasked to <fix, fire> (e.g., UAS1) is not directly connected to the TL, and therefore receives the command by relay from another UAS (e.g., UAS2). However, because UAS2 does not have authority over UAS1, UAS1 rejects the command as invalid.
  - The UAS tasked to <fix, fire, search> (e.g., UAS1) is momentarily disconnected from the team. This triggers a change in the automation operating modes that removes the UAS from the team (see Dynamic Membership) until it is explicitly commanded to rejoin the team.

S-37.1.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. *The context of this UCCA does not apply to this top-level scenario.*

***Note**: The UCCA assumes that it is necessary to provide tasks.*

S-37.1.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. *The context of this UCCA does not apply to this top-level scenario.*

***Note**: N/A because the TL cannot provide the fix command, as specified in the system assumptions.*

S-37.1.9 (Step 4: Other Factors): The controllers on the team do not provide any commands due to factors beyond those explored in interactions between the controllers on the team.

- *Unsafe Feedback Path:* see S-37.1.3

[24] (B) Scenario: An attacker causes a mode change to a friendly flight mode that eliminates the ability to provide "fix on target" commands.

- *Unsafe Control Path:* see S-37.1.3
- *Unsafe Process Behavior:* the UAS provides a <fix, fire>, but the target deploys countermeasures or performs defensive maneuvers that overcome its effects.

UCCA 38.2 (abstracted UCCA): Controller Ci provides fix and no other Cj provides fix when Ci is not capable and a Cj is [H3] [DA]

> **UCCA 38.2.1 (refined UCCA)**: UAS1 provides fix and UAS2 does not provide fix when UAS1 is incapable and UAS2 is.

The figure below shows the relevant internal control combinations that can lead to the collective output in the UCCA. Each case is traced to the relevant high-level scenarios using labels. Cases 1a and 1d involve multiple unsafe actions that are less logical together. In 1a, the TL does not task any UAS to fix, but an untasked UAS provides the fix, and that UAS is the "wrong UAS" for the task. In 1d, the TL tasks multiple UAS to fix, but only one of those UAS provide fix, and that UAS is the "wrong UAS" for the task. These contrived combinations are skipped in the analysis.

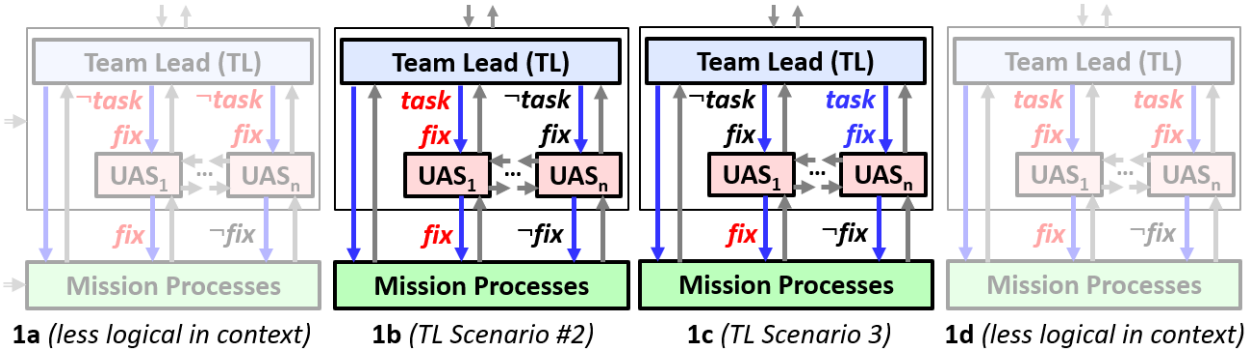


Figure A3-5. Internal control combinations that result in the wrong UAS providing a fix

S-38.2.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. *The UCCA context does not apply to this top-level scenario.*

> ***Note:*** *the scenario could reference instances when (1) the TL does not task UAS to fix and (2) only an incorrect UAS provides a fix. Item (1) is analyzed in the context of S-37.1.1. Item (1) and (2) are less logical together.*

S-38.2.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, TL tasks a UAS to fix, which cannot provide the fix.[25] [DA]

> ***Note:*** *the scenario could also include the TL tasking multiple UAS to fix, as addressed in S-37.1.2.*

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL interprets direction from higher authorities that a specific controller <TL, UAS1, UAS2> must be used to provide a specific command <fix, fire, search>, even if it is not capable of doing so. TL could also interpret direction that a specific controller must not be used to provide a specific command. ***Refinement:*** similar to S-37.1.1.

[25] (B) UCA 2: Team Lead provides "fix on target" command to the wrong UAV (H3)

- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has one or more of the following inadequate process model variables: (1) TL believes a specific controller <TL, UAS1, UAS2> can perform the <fix, fire, search> task, (2) TL does not believe a specific controller <TL, UAS1, UAS2> can perform the <fix, fire, search> task. ***Refinement:***
    - *Missing Feedback*: The feedback does not provide information regarding the capability of each UAS to execute a task.
    - *Missing Feedback*: The feedback does not provide information regarding if the UAS mode of operation of the automation allows it to take on mission tasks. The UAS may be in a mode that only accepts basic waypoints and not higher-level tasks.
    - *Inadequate feedback.* The interface that displays the UAS capabilities is difficult to access or read in a high workload environment.
    - *Delayed Feedback.* UAS1 loses the ability to provide a <fix, fire, search> command (e.g., equipment malfunction, low fuel). However, communication degradations (e.g., jammed links) prevent a timely status update from reaching TL.
    - *Feedback not sent.* UAS2 is temporarily disconnected. The TL is not aware it is capable of providing a fix.
    - …
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the team capabilities, but still chooses to task UAS1 to provide the <fix, fire> command. ***Refinement:***
    - TL misunderstands how the UAS operate. He believes the UAS will automatically change the assignment to a UAS that is capable.
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task UAS2 but inadvertently tasks UAS1. This can occur, for example, if the tasking interface design does not guard against such slips given the workload and operating environment.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*****:** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*****:** *Not applicable. Top-level scenario focuses on control decisions from TL only.* (See Inadequate Process Model / Control Algorithm)
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** UAS1 was recently added to the team. Until then, all the UAS working with the TL were capable of providing the fix command. Therefore, the TL was not mentally tracking the possibility that a UAS would not be. UAS1 is added, and TL tasks it to fix without considering its ability to do so.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** No scenario conceived.

S-38.2.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks UAS2 to fix, but UAS1 provides the fix instead, even though it is not capable to do so[26]. [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:***

[26] (B) UCA 30: Autonomous Controller implements a task using the wrong UAV (H3)

- UAS1 is tasked to <fix, fire> by another controller (e.g., ground station, another team lead, or a cyber attacker), which causes UAS2 to drop its task. (Similar to S-37.1.3)

- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:***
  - UAS1 does not intend to <fix, fire>. However, its <targeting, weapon system> equipment malfunctions and it releases <energy, weapon> anyways. UAS2, which is assigned the task, observes this action, assumes UAS1 has taken on that task, and drops its own plans to provide it.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same coupling as in S-37.1.3 (Figure A3-2).
  - ***Feedback about Controlled Process from Collaborators*:** Before UAS2 starts to fix, the controller assigned to fire provides feedback to the team that the fix on the target is inadequate to fire. UAS1 receives this feedback and misinterprets it as a request for it to provide a fix. UAS1, therefore, claims the task, even though it lacks the capability for it, and UAS2 drops its task to avoid creating interferences.
  - ***Feedback about Collaborator Control Actions from Controlled Process***: Same as S-37.1.3. But leads to the wrong UAS taking on the task and the correct one dropping it.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** Same illustration as in S-37.1.3 (Figure A3-3).
  - ***Construction*:** The control algorithms on the different UAS are not compatible with one another. UAS1 (tasked to fire), has a control algorithm that assumes it must also provide the fix. Therefore, it claims the fix task, and UAS2 drops it. This could be due to a misconfiguration.
  - ***Initialization*:** Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement:***
    - The different UAS have different beliefs about the current roles and responsibilities of other controllers on the team. Examples:
      - One UAS believes that UAS1 has taken on the <fix, fire> task based on stale information. It disseminates this belief to the rest of the team. As a result, UAS2 loses confidence in its assignment, drops the <fix, fire> task, and UAS1 takes it on. (Similar to S-37.1.3)
      - TL and UAS1 have different models of which controllers are active in the team (related to Dynamic Membership). UAS1 believes it is the only UAS that can be tasked on the team. It incorrectly interprets the tasking TL provides to UAS2 as intended for UAS1.[27]
  - ***Model Updates*:** Elements of the process models are not adequately or consistently updated across the team. ***Refinement:***

[27] (B) Scenario: AuC cannot identify the selected UAV and chooses an alternate UAV based on optimal firing position and weapon type

        - *Vertical Coordination (Control)*: No scenario conceived.
        - *Lateral Coordination (Communication):* UAS1 receives information from a teammate that leads it to believe it was tasked by the TL to provide the <fix, fire> command. Thus, it takes on the task and <TL, UAS2> drops it. For example:
            - A UAS tasked to <fire, fix> has stale information that leads it to collaborate with UAS1 to engage the target. That UAS incorrectly starts coordinating the joint-task with UAS1. This prompts UAS1 to believe it has been assigned to provide the fix.
            - A UAS is disconnected from the team and has stale information on the current designation of roles and responsibilities. Once it reconnects, its shares its understanding of the delegation of authorities, which includes UAS1 providing the <fix, fire> command. This prompts UAS1 to believe it has been assigned to provide the <fix, fire> command.
        - *Lateral Coordination (Observation)*: Same scenario as S-37.1.3, but leads to wrong UAS taking on the task and the correct one dropping it.
        - *Prediction*: No scenario conceived.
    - ***Decision-Making*:** Due to reasons listed above, the properly tasked UAS2 drops its task. The UAS team is programmed with a heuristic to force the assignment to the UAS with, for example, the lowest ID number (UAS1). UAS1 takes on the <fix, fire, search> task, without considering its capability or suitability to do it.[28]
    - ***Capacity***: UAS1 has an internal process malfunction and is incapable of processing new task assignments from the TL. Its roles and responsibilities are "stuck" on those from the previous plan. As a result, UAS1 continues to execute the <fix, fire> command, which prevents the correctly assigned UAS from executing it.[29]
- ***Dynamic Membership:***
    - UAS1, previously tasked by TL to provide the fire command, is taken offline due to malfunction(s) that impact its ability to fulfill mission tasks such as <fix, fire>. However, it temporarily rejoins the team, after the TL re-tasks the command to UAS2, and shares its intent to fulfill that original task with the team. This prompts UAS2 to drop its tasks.
    - UAS1 was offline as the TL was formulating the plan to task UAS2 to provide the <fix, fire> command. UAS1 suddenly becomes online just before the TL issues tasking. The TL inadvertently tasks UAS1 to instead of UAS2.
- ***Dynamic Connectivity:***

[28] (B) Scenario: AuC control algorithm changes UAV to select the most optimal firing position; AuC calculates that it is more optimal for another UAV to implement task so it diverts the task to other UAV
[29] (B) Scenario: AuC implements a delayed command from TL when a different UAV was selected for the target

- UAS2 is indirectly connected to TL via UAS1 as a relay. When TL tasks UAS2 to <fix, fire> via UAS1, UAS1 misinterprets the message as it being assigned the task. As a result, UAS1 provides the <fix, fire>, and UAS2 does not.

S-38.2.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. *The context of this UCCA does not apply to this top-level scenario.*

***Note**: The UCCA assumes that it is necessary to fix, as such there is no adequate way to not provide the fix task.*

S-38.2.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. *The context of this UCCA does not apply to this top-level scenario.*

***Note**: The TL does not have the option to provide the fix command by system definition.*

S-38.2.9 (Step 4: Other Factors): The "wrong" UAS provides a fix due to factors beyond those explored in interactions between the controllers on the team. [DA]

- *Inadequate Process Feedback:* The feedback the team senses from the target leads controllers on the team to believe UAS1 is better suited to provide the fix task than UAS2. For example, it is incorrectly believed the target will be better fixed in a wavelength available to UAS1 and not UAS2.
- *Unsafe Process Behavior:* The target employs countermeasures that cause the team to believe UAS1 has more adequate capabilities to provide the <fix, fire> task.

UCCA 39.3 (abstracted UCCA): Controller Ci provides fix and another Cj provides fix when that creates mutual interference [H1,3] [DA]

UCCA 39.3.1 (refined UCCA): UAS1 provides fix and UAS2 provides fix

The figure below shows the relevant internal control actions that can lead to the collective UCCA output and traces them to the top-level scenarios. Case 1a is contrived, in that the TL does not task any UAS to fix and multiple UAS provide a fix in a way that interferes with each other. It is skipped from the analysis.

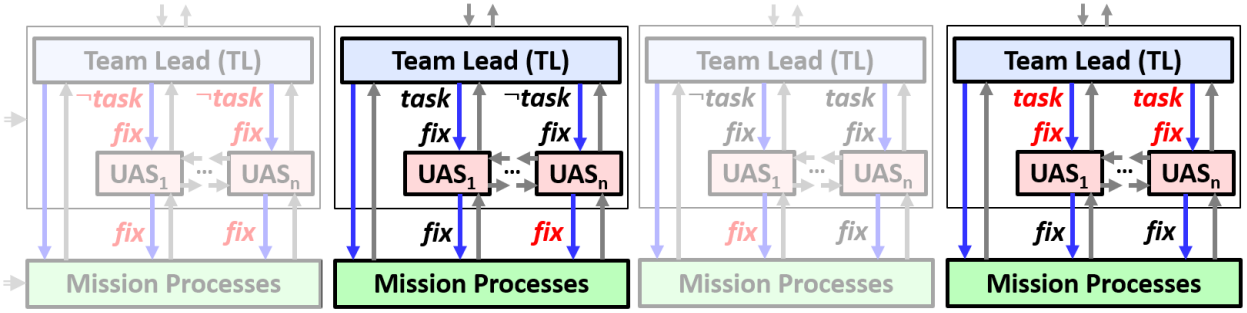


Figure A3-6. Internal control combinations that result in the wrong UAS providing a fix

S-39.3.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. *The context of this UCCA does not apply to this top-level scenario.*

> ***Note**: the scenario could reference instances when (1) the TL does not task UAS to fix and (2) multiple UAS provide a fix. However, items (1) and (2) are less logical together.*

S-39.3.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, TL tasks UAS1 and UAS2 to fix. As a result, the multiple UAS provide a fix and interfere with each other. [DA]

- *All factors:* Same as S-37.1.2

S-39.3.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks UAS2 to fix, but UAS1 provides a fix in addition. [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-38.2.3
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same S-38.2.3, but leads to two UAS executing the task.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same coupling as in S-37.1.3 (Figure A3-2).
    - *Feedback about Controlled Process from Collaborators*: Same scenario as S-38.2.3, but leads to two UAS providing a fix.
    - *Feedback about Collaborator Control Actions from Controlled Process*: Same scenario as S-37.1.3, but leads to two UAS providing a fix.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** as illustrated in Figure A3-3.
    - Construction, Initialization, Model Updates, and Capacity: Same scenario as S-38.2.3, but leads to two UAS providing a fix.
    - Decision-Making: The UAS team is programmed with a heuristic to force the assignment of the task to a UAS in case there is conflict. However, in some corner cases, the heuristic leads to two different UAS being assigned. (Similar to S-38.2.3)
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.2
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.2

S-39.3.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. *The context of this UCCA does not apply to this top-level scenario.* (Same as S-38.2.4)

S-39.3.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. *The context of this UCCA does not apply to this top-level scenario.* (Same as S-38.2.5)

S-39.3.9 (Step 4: Other Factors): Multiple UAS produce a fix due to factors beyond those explored in interactions between the controllers on the team. [DA]

- *Inadequate Process Feedback:* Shortly after being tasked to fix, UAS1 loses adequate feedback of the target, and reports this to the TL. Therefore, TL tasks UAS2 to provide the fix, but does not untask UAS1. UAS1 then regains sufficient feedback and provides a fix.
- *Unsafe Control Path to Process:* Same as S-38.2.3, but leads to two UAS executing the task.
- *Unsafe Process Behavior:* No scenario conceived.

UCCA 40.4 (abstracted UCCA): Controller Ci does not fire and no other Cj fires when there is a priority target to engage and a teammate able to fix [H3] [DA]

UCCA 40.4.1 (refined): TL does not fire, UAS1 does not fire, and UAS2 does not fire

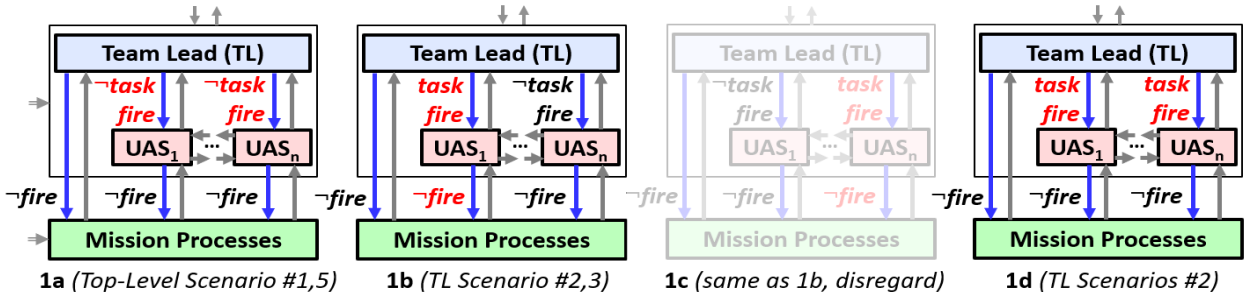


Figure A3-7. Internal control combinations that can contribute to no controller firing

S-40.4.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, TL does not task any UAS to fire (and does not intend to fire her/himself). As a result, no fire command is provided[30]. [DA]

- ***All factors:*** Same as S-37.1.1

S-40.4.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, (1) TL tasks multiple UAS to fire, or (2) the TL tasks a UAS to fire that cannot fire. [DA]

Note: *because (2) is emphasized in the context of UCCA 41.5, this scenario focuses on item (1).*

- ***All factors:*** Same as S-37.1.2

S-40.4.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks a UAS to fire, but the tasked UAS does not fire.[31] [DA]

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Similar to S-40.4.5, but in Machine-Machine interactions
- ***All other factors:*** Same as S-37.1.3

[30] (B) UCA 17: Team Lead does not provide fire command when the mission requires it (H3)
[31] (B) UCA 35: Autonomous Controller does not release weapon when the Team Lead commands it (H3)

S-40.4.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. *The context of this UCCA does not apply to this top-level scenario.* Same as S-37.1.4

S-40.4.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, the TL does not fire (and does not task a UAS to fire). Emphasis is placed both on when the TL does not intend to fire, and when s/he does intend to fire but does not do so[32]. [DA]

> ***Note:*** *The fire command is coupled to the fix command. In this system, only UAS (machines) provide the fix command. If the TL (human) intends to provide the fire command, as in this UCCA, scenario analysis focuses on human-machine collaboration.*

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.1
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-37.1.1. **In addition**: TL does not believe there is a team member (including her/himself) available to task the fire. ***Refinement***:
    - *Feedback not available.* TL is fixated on tasking UAS and forgets that s/he can fire on the target her/himself. The feedback describes the set of UAS resources available, but it does not append the resources the TL can contribute to the tasks.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the target and the team capabilities but still chooses not to fire or task a UAS to fire. ***Refinement:***
    - TL is currently busy and prioritizing other operating tasks, but intends to fire. However, s/he later forgets to do this due to heavy workload. (Similar to S-37.1.1)
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.1
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** see Figure A3-2.
    - ***Feedback about Controlled Process from Collaborators*:**
        - Prior to firing, TL misinterprets feedback from a UAS that the target being fixed was destroyed and there is no need to fire. ***Refinement:***
            - The UAS broadcasts feedback for a fix of a different target, for a different teammate to fire on. (Similar to S-37.1.3)
            - The UAS actually provided feedback that the target was not destroyed, but the TL interface is inadequate and leads to misinterpretation, especially in high workload.
            - (Human-Machine) Prior to the TL firing, the UAS provides an alert to the TL indicating the target is ready to fire on. However, the alert is too vigorous in tone and is similar to how a human wingman would advise **NOT** to fire. TL misinterprets it and does not fire.
            - (Human-Machine) The UAS provides feedback that is semantically misaligned with how the TL interprets it. For instance, the UAS labels a damaged target as destroyed, even though the target is not disabled and should still be fired on.

[32] (B) UCA 17: Team Lead does not provide fire command when the mission requires it (H3)

- ***Feedback about Collaborator Control Actions from Controlled Process*:**
  - TL does not receive the fix feedback necessary to close the fire loop that should be provided by a collaborating UAS. Similarly, the UAS does not receive feedback from the TL that the fix it is providing is inadequate.[33] ***Refinement***:
    - Coordination between TL and UAS leading up to execution is flawed (e.g., reasons include incorrect, delayed, misinterpreted, and missing coordination). Specifically, coordination is hindered by inherent differences in human and machine interpretation of the joint task.
    - The fix signal received by the TL from the UAS is degraded (e.g., malfunction, lack of compatibility, interference).
    - Similarly, the feedback provided by the TL regarding the degraded fix signal is not adequately received by the UAS.
  - The TL receives fix feedback that is not consistent with the target it was expecting to fire on, and therefore chooses not to fire. For example, it the feedback that is more aligned with a friendly target.[34] ***Refinement***:
    - This signature or location of the fix is similar to that of another (friendly) target.

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
  - ***Construction*:**
    - (Human-Machine) The control algorithms on the UAS are not compatible with how the TL specifies the <fix, fire> task, which inhibits collaborative control in coupled tasks. (Same as S-37.1.3). For example, their general strategy to execute the task is different due to differences in how the TL was trained and the version of the autonomy programmed into the UAS.
  - ***Initialization*:**
    - (Human-Machine) The TL lacks awareness of how task parameters provided to a UAS contribute to its execution plan for a fix task. As such, the TL cannot fully anticipate how the UAS will behave to provide the fix that supports the TL fire command. This hinders implicit coordination and prevents the controllers from executing the coupled fix-fire task.
    - (Human-Machine) The fix task parameters provided to the UAS by the TL (coupled with TL fire command) do not include multiple subconscious variables tracked by the TL to execute the task. These variables are founded on creativity and past experience from the TL. As a result, the TL and UAS start with an inherently different set of assumptions for their collaborative task. This hinders effective coordination during execution and ultimately prevents them from executing the joint fix-fire task.

---

[33] (B) Scenario: Autonomous Controller receives feedback about the target, but cannot identify the target designation, so AuC releases the missile for another target.

[34] (B) Scenario: AuC algorithm determines the target the TL is locked on is a friendly or civilian target, so AuC does not implement the command

- ***Model Updates*:**
  - *Vertical Coordination (Control)*: TL micromanages the activities of the UAS tasked to fix (e.g., UAS1) so that they are coordinated with the TL fire command. However, the ability to manage both the UAS and the TL's own aircraft exceeds her/his workload capacity. The TL makes mistakes (e.g., inconsistent information provided, omissions, …) and the UAS does not issue the necessary fix, so the TL does not fire.[35]
  - *Lateral Coordination (Communication)*:
    - (Human-Machine) Lateral coordination between UAS1 (tasked to fix) and TL (self-tasked to fire) is hindered by human-machine asymmetry in information semantics and timing. (Same as S-37.1.3)
    - There is a lack of consensus on the joint process to control. During execution, a UAS expresses differences in the target to engage, where, or when to do it. The TL anticipates this will confuse the UAS providing the fix, loses confidence it will do so, and thus decides not to fire. For example:
      - A teammate shares general information about a different target it is tasked to engage (Similar to S-37.1.3).
      - A teammate was temporarily disconnected and its model of the target diverged from that of the team. (Similar to S-37.1.3)
  - *Lateral Coordination (Observation)*: Same as S-37.1.3, in addition:
    - (Human-Machine) The information the TL gains by observing a collaborating UAS lacks implicit coordination information that is typically provided by a human teammate (e.g., tone of voice, reaction time, demeanor, …). Similarly, this type of information, which the TL subconsciously outputs to communicate with humans, is not recognized by the UAS.
  - *Prediction*: Same as S-37.1.3, but with roles reversed (TL fires, UAS fixes).
    - (Human-Machine) Human and Machine training experiences have evolved separately. The TL, which previously trained with humans, can anticipate when a human teammate will provide a fix based on implicit observations and messages. Similarly, the machines may have been trained using simulation data, which does not precisely recreate human behavior. As a result, both controllers lack context or misinterpret certain cues that affect when and how to provide their respective commands (Similar to Observation).
- ***Decision-Making*:**

[35] (B) Scenario: Team Lead is pre-occupied with another task so provides command late

        - Despite adequate communication channels, the distributed decision-making process is too slow to keep up with the dynamic state of shared-controlled process. (Same as S-37.1.3)
        - The TL has low confidence that the collaborating UAS will provide a reliable fix at the time that s/he fires. Thus, the TL precautionarily does not fire. Reduced confidence may occur due to the flawed coordination factors listed above. The TL may also have previously received a faded fix signal from the UAS that suggested degraded reliability of the system.
    - ***Capacity*:** Same as S-37.1.3, in addition:
        - The TL lacks the cognitive capability to direct the joint process execution. This may be due to excessive workload requirements imposed by the mission complexity, an inadequate system design, and/or insufficient experience in a relevant environment.[36]
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - TL anticipates that the UAS tasked to fix may exit the team (e.g., retasked by Ground Station or attached to a different team). As such TL holds off on firing.
    - A new UAS is added to the team that is better suited to issue the fire command. The TL now intends to task that UAS, but does not do so due to factors described in S-37.1.1.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
    - During execution, the TL becomes only able to communicate with the UAS intended to fix (e.g., UAS1) indirectly, through another UAS (e.g., UAS2). However, the TL anticipates that UAS2 will relocate for other tasks, which may lead to a disconnect with UAS1. The TL holds off on firing.

S-40.4.9 (Step 4: Other Factors): No member of the team fires due to factors beyond those explored in interactions between the controllers on the team. [DA]

- Same as S-37.1.9

UCCA 41.5 (abstracted UCCA): Controller Ci fires and no other Cj fires when Ci is not capable and a Cj is [H3] [DA]

UCCA 41.5.1 (refined UCCA): TL fires, UAS1 does not fire, and UAS2 does not fire when TL is not capable and UAS1 or UAS2 is.

UCCA 41.5.2 (refined UCCA): TL does not fire, UAS1 fires, and UAS2 does not fire when UAS1 is not capable and TL or UAS2 is.

The figure shows the relevant internal control actions and traces them to top-level scenarios. Cases 1d and 2d are contrived and are not analyzed for similar reasons as in previous UCCAs.

[36] (B) Scenario: Team Lead is pre-occupied with another task so provides command late

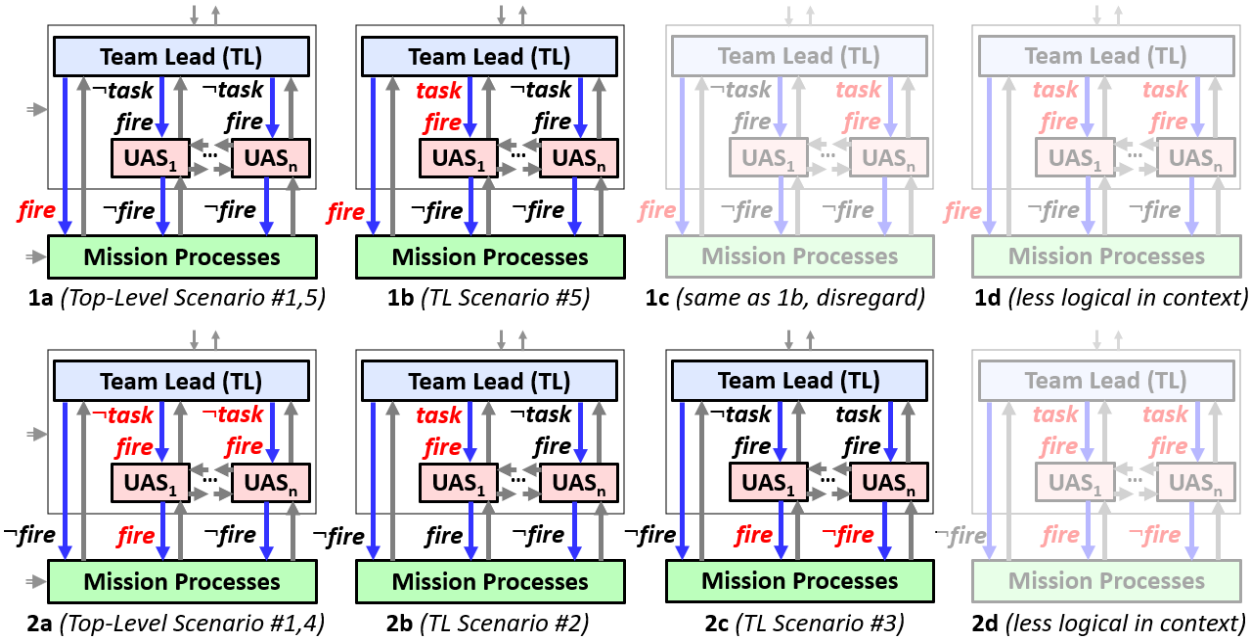


Figure A3-8. Internal control combinations that result in the wrong UAS providing fire

S-41.5.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. *The context of this UCCA does not apply to this top-level scenario.* (Similar to S-38.2.1)

S-41.5.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, TL tasks a UAS to fire, which cannot provide the fire.[37] [DA]

- *All factors:* Same as S-38.2.2

S-41.5.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks UAS2 to fire, but UAS1 provides the fire instead, even though it is not capable to do so.[38] [DA]

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*:**
  - ***Feedback about Controlled Process from Collaborators*:** Same as S-38.2.3. But leads to the wrong UAS taking on the fire task.
  - ***Feedback about Collaborator Control Actions from Controlled Process***: Same as S-37.1.3. But leads to the wrong UAS taking on the task and the correct one dropping it.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*:** Same as S-38.2.3
- ***All factors:*** Same as S-38.2.3 and S-41.5.5

S-41.5.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective

[37](B) UCA 19: TL provides firing command for the wrong UAV (incorrect weapon target pairing) (H3)
[38](B) UCA 30: Autonomous Controller implements a task using the wrong UAV (H3)

control. Here, TL intends to fire or tasks a UAS to fire, but instead an untasked UAS without adequate capability fires.[39] [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-38.2.3
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-38.2.3
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Before <TL, UAS2> can fire UAS1 misinterprets the following factors as a request for it to fire on the target. Therefore, it claims the task and broadcasts its intent to the team. As a result, <TL, UAS2> does not provide its fire command.
    - ***Feedback about Controlled Process from Collaborators*****:** UAS1 receives general feedback from the UAS providing the fix that the target is not destroyed.
    - ***Feedback about Collaborator Control Actions from Controlled Process*****:** UAS1 receives the targeting energy provided by another UAS's fix.[40]
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** as illustrated in Figure A3-3.
    - ***Construction*****:** Same as S-38.2.3, but leads UAS1 to claim the fire task.
    - ***Initialization*****:** No scenario conceived.
    - ***Model Updates*****:** Same as S-38.2.3. **In addition**:
        - *Lateral Coordination (Observation)*: TL observes UAS1 maneuvering in a way consistent with providing a fire command. TL loses confidence in her/his understanding that UAS1 cannot fire and misinterprets this observed behavior as an ability to fire tasked by another controller (Ground Station). As a result, the TL (reluctantly) retasks UAS1 to fire to match the observed behavior of the system.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-38.2.3
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** UAS1 is disconnected from the team. In such circumstances, it switches to a control mode that authorizes it to fire under certain circumstances (e.g., aware that a teammate was planning tasked to fire, in self-defense, …). Such modes are built into the system to enable continuity of operation should the team become disconnected or suffer severe losses. However, in this case, UAS1 takes on the task even though it is not capable, and TL drops it in observation of this.

S-41.5.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL fires her/himself when they are not capable to, instead of tasking a UAS that can. [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.2 and S-38.2.2
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-38.2.2. ***Refinement:***

---

[39](B) UCA 31: Autonomous Controller implements a task when there is no command from an authorized command provider (H3)

[40] (B) Scenario: AuC receives a target designation that it determines can lead to releasing a missile as long as parameters are satisfied, so AuC decides to release the weapon without a Team Lead command

- *Inadequate feedback:* TL interface hides the TL's own aircraft status and capabilities behind that of the UAS (or vice versa) and makes it difficult to consider all resource options available on the team.

- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See Cognitive Alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:***
  - TL has a weapons system malfunction and fires unintentionally, even if it is not configured appropriately to strike the target. The UAS assigned to fire observes this and drops its task[41].
  - TL intends to task a capable UAS to fire but gets lost in the user interface and is not capable of doing so. Out of frustration or desperation, TL fire her/himself.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Before a tasked UAS can fire, TL misinterprets the following factors as an indication that the UAS tasked to fire is not executing its task. This contributes to the TL attempting to fire her/himself even if s/he is unable: (Similar to S-41.5.4)
  - ***Feedback about Controlled Process from Collaborators*:** TL receives general feedback from the UAS providing the fix that the target is not destroyed. (Similar to S-41.5.4)
  - ***Feedback about Collaborator Control Actions from Controlled Process*:** TL receives the targeting energy provided by another UAS's fix. (Similar to S-41.5.4)
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. Refined:
  - ***Construction***:
    - (Human-Machine) The control algorithms on the UAS are not configured to respond in the way that the TL expects. For example, the UAS do not acknowledge the <fire, search> task, and do not provide feedback of <firing parameters, search area>, … This is different from the TL's previous interactions with a MUM-T system. As a result, the TL loses confidence in the ability of the UAS to execute the <fire, search> command, fixates on executing it, and takes it on her/himself.
  - ***Initialization***:
    - (Human-Machine) The <fire, search> task parameters provided to the UAS by the TL do not include multiple subconscious variables tracked by the TL. As a result, the response of the UAS team to the task is unsatisfactory to the TL. S/he cancels the tasking and executes the <fire, search> command her/himself. (Similar to S-40.4.5)
  - ***Model Updates*:** Elements of the UAS's process models are not adequately or consistently updated. Refined:
    - *Vertical Coordination (Control)*: TL prescribes certain coordination parameters associated with the fix-fire tasks that are assigned to UAS. These parameters are typically determined by the controllers assigned to

[41] (B) Scenario: The UAS [or any controller] does not intentionally release a missile, but there is electrical interference that causes the bay to unlock, or a missile is loose in the bay, so a missile is released.

the tasks. As a result, the UAS tasked to fire incorrectly believes that the TL will fire instead, and therefore, drops its task. This possibly repeats its self with other UAS. Because the TL is unable to provide these coordination parameters with the UAS believing it has taken on the fire task, the TL decides to take on the task her/himself.

        - *Lateral Coordination (Communication)*: For the reason above (vertical coordination), some of the UAS incorrectly believe the TL has taken on the <fire, search> task. They communicate this belief to the rest of the team, which causes the assigned UAS to drop the task. Because the TL is unable to provide these coordination parameters with the UAS believing it has taken on the fire task, the TL decides to take on the task her/himself.
        - *Lateral Coordination (Observation)*: The TL loses the ability to observe the UAS tasked to <fire, search> and is no longer confident the task is being executed. As a result, the TL decides to take on the search task her/himself, even if in addition to other UAS executing it.
        - *Prediction*: Based on current observations, the TL does not believe a UAS was successfully tasked with the <fire, search> task, nor will one be able to be tasked with it or perform it (anti-access environment). As a result, the TL takes on the task her/himself.
    - ***Decision-Making*:** See all the model update scenarios above.
    - ***Capacity*:** Same as S-40.4.5
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** The set of active UAS changes so often that the TL has no confidence that tasking the UAS team will result in an assigned UAS that actually completes the task. S/he chooses to take on the task her/himself.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S-41.5.9 (Step 4: Other Factors): The "wrong" controller provides the fire command due to factors beyond those explored in interactions between the controllers on the team. [DA]

- *No new scenarios conceived.*

UCCA 42.6 (abstracted UCCA): Controller Ci fires and another Cj fires when that creates excessive damage [H3, H5] [DA]

UCCA 42.6.1 (refined UCCA): TL provides fire and UAS1 provides fire

UCCA 42.6.2 (refined UCCA): UAS1 provides fire and UAS2 provides fire

Another UCCA listed in Table A2-5 leads to the same outcome but is lower in priority. UCCA 42.6.3 consists of every controller providing fire, which has similar factors to the analysis below.

The figure below shows how internal control combinations relevant to the UCCA trace to top-level scenarios. Cases 1d and 2d are not analyzed for similar reasons as in previous UCCAs.

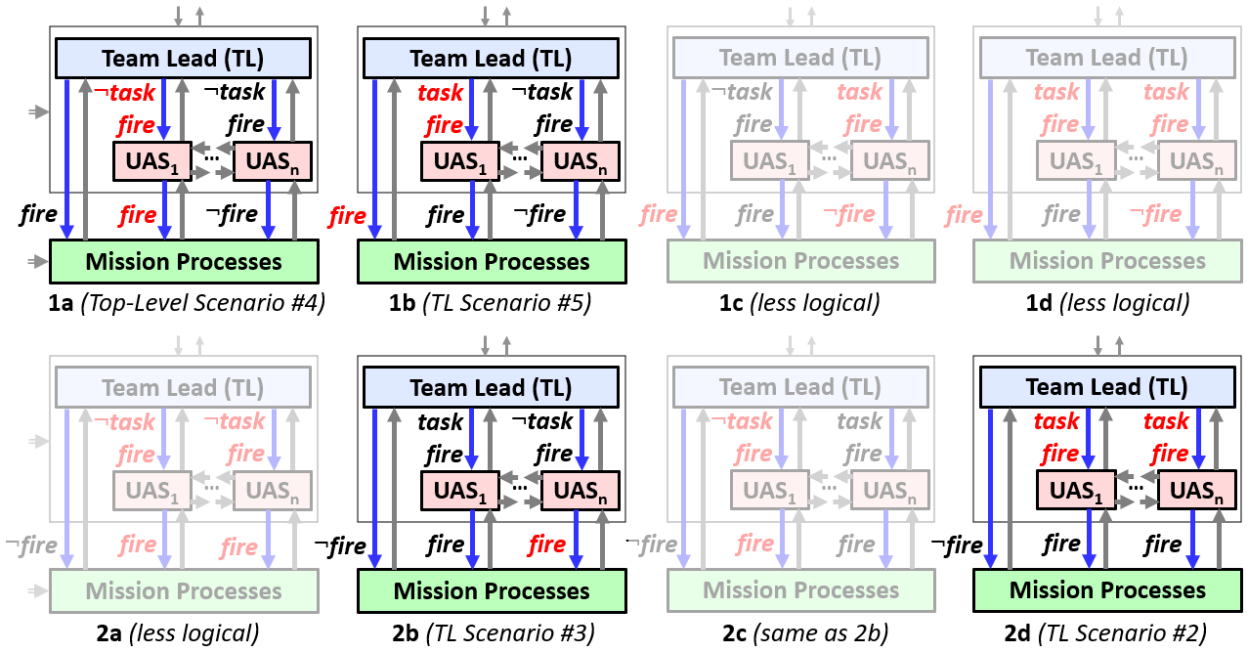


Figure A3-9. Internal control combinations that can lead to multiple controllers firing

S-42.6.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. *The context of this UCCA does not apply to this top-level scenario.*

S-42.6.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, TL tasks UAS1 to fire and tasks UAS2 to fire, so both fire. [DA]

- ***All factors:*** Same as S-37.1.2

S-42.6.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks a single UAS appropriately to fire, but multiple UAS perform the task. [DA]

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-41.5.4.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment, Dynamic Membership, Dynamic Connectivity:*** Same as S-39.3.3.

Note: *S-42.6.3 is nearly identical to S-39.3.3. Both involve a machine tasked and a machine untasked that execute a common command <fix, fire>, which leads to a hazard. As such, their causal factors are nearly identical, with the exception of details on of how control loops are mutually closed.*

S-42.6.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, TL fires as intended, and an untasked UAS fires in addition.[42] [DA]

[42] (B) UCA 31: Autonomous Controller implements a task when there is no command from an authorized command provider (H3)

- ***All factors:*** Same as S-41.5.4, but leads to TL and UAS firing.

S-42.6.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL provides fire and tasks UAS1 to fire. [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.2
- ***(Step 2: Internal Control) Inadequate Process Model:*** Similar to S-37.1.2, with further ***refinement*:**
    - *Delayed feedback.* TL tasks UAS1 to <fix, fire>, but feedback from UAS1 acknowledging the tasking is delayed back to the TL (e.g., degraded communication channel, hidden in layers of the interface). Thus, TL incorrectly assumes UAS1 was not tasked and decides to execute the task her/himself.
    - *Inadequate feedback.* TL has inadequate feedback of the target to make the determination that multiple controllers firing will create excessive damage. As such, TL elects to conservatively assign multiple controllers to fire.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the target and the team, but still chooses to task multiple UAS to fire. ***Refinement:***
    - TL has previously experienced weapon system failures on the team. This leads her/him to believe that assigning multiple controllers to fire (including her/himself) is necessary to ensure at least one weapon is successfully fired. However, here, all weapon systems function and fire together.
- ***(Step 2: Internal Control) Unsafe Control Path:***
    - TL intends to provide the fire command and not task a UAS, but inadvertently tasks a UAS to fire in addition. ***Refinement:***
        - The tasking interface is inadequate given the workload and operating environment. (Same as S-37.1.2)
        - TL had originally tasks UAS1 to <fix, fire> but changes the plan, and instead plans to fire. However, the communication channel to UAS1 is degraded and it does not receive the task cancellation.[43]
    - TL tasks a UAS to fire, but s/he inadvertently fires in addition. ***Refinement:***
        - The tasking interface is inadequate given the workload and operating environment. (Same as S-37.1.2)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-41.5.5, but leads to TL firing in addition to a UAS.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** Same as S-41.5.5, but leads to TL firing in addition to a UAS.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.2
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** No scenario conceived.

S-42.6.9 (Step 4: Other Factors): Multiple controllers provide the fire command due to factors beyond those explored in interactions between the controllers on the team. [DA]

---

[43] (S) Scenario: FMS has a fix on target command with authorization, there is a delay in another command from the Team Lead to delete the command, so AuC receives a fix command without authorization

- Same as S-39.3.9

UCCA 43.7 (abstracted UCCA): Controller Ci does not provide search and no other Cj provides search when no targets have been found [H3] [DA]

UCCA 43.7.1 (refined UCCA): TL, UAS1, and UAS2 do not provide search command

The figure below shows the relevant internal control actions for the UCCA. The way the TL issues a search task is different than how s/he issues the fix, fire tasks. The search task is provided to the UAS team, which allocates it to a UAS. The figure below show the "Task Search" command going to the overall set of UAS to highlight this distinction in the analysis that follows.

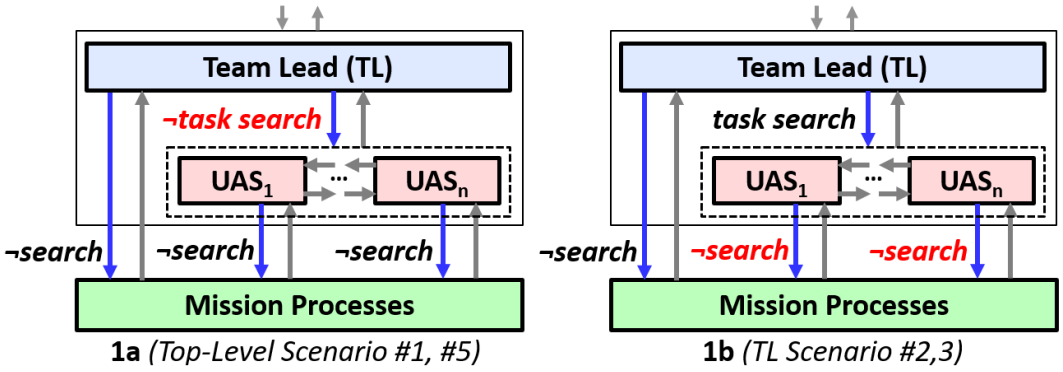


Figure A3-10. Internal control combinations to consider that can lead to no controller searching.

S-43.7.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, TL does not task the UAS team to search (and does not intend to search her/himself)[44]. [DA]

- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-37.1.1, in addition:
  - *Inadequate Feedback.* Mission feedback provided to TL is stale and shows an outdated target location. The TL believes s/he has an accurate target location, and therefore does not need to search.
- ***All other factors:*** Same as S-37.1.1

S-43.7.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control.

> ***Note:*** *for the search task, tasking multiple UAS is not unsafe. Scenarios in which the TL tasks the UAS to search in a way that contributes to the UAS not searching are covered in S-43.7.3*

S-43.7.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, TL tasks the UAS team to search, but no UAS searches.[45] [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.3

---

[44] (B) UCA 8: Team Lead does not provide identify target command when the mission requires a determination of the objects in a region (H3)

[45] (B) UCA 29: Autonomous Controller does not implement a task from the Team Lead (H3)

- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-37.1.3
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.3
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** If the search tasks are statically defined per planning cycle, then this collaborative dynamic does not apply. However, if the UAS exchange and use information gathered during searches to dynamically update their model of the remaining task, then this dynamic applies, as illustrated in Figure A3-11. ***Refinement:***

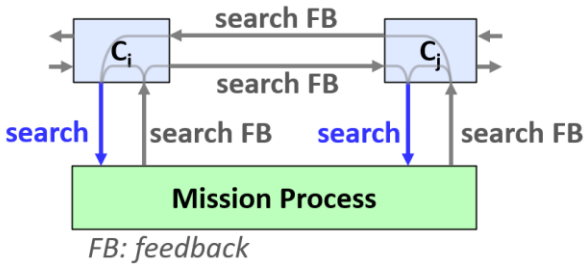


Figure A3-11. Control structure for controllers collaborating on a search task

  - ***Feedback about Controlled Process from Collaborators*:** UAS1 is selected by the automation to complete the search task. UAS2 is in the vicinity of the search area and points its search sensor to gather feedback of opportunity. UAS1 incorrectly uses that feedback to update its belief that the search task is no longer necessary, and therefore drops the task, even though it is still needed.
  - ***Feedback about Collaborator Control Actions from Controlled Process*:** N/A
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The UAS on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
  - ***Construction***: The process models and/or control algorithms of the UAS are not adequately or consistently built to support collaborative control. ***Refinement***:
    - The control algorithms on the UAS are not compatible with how the TL specifies the search task. This prevents the task allocation process from initializing This could occur due to a misconfiguration of the UAS, the TL software, or degrees of freedom available to the TL in defining search tasks that go beyond what the UAS can handle. (Similar to S-37.1.3)
    - The control algorithms on the different UAS are not compatible with one another (e.g., misconfiguration). As a result, the UAS lack the ability to coordinate and select who will take on the task. (Similar to S-37.1.3)
  - ***Initialization***: Elements of the UAS's process models are not adequately or consistently initialized across the team. ***Refinement***:
    - The UAS receive different versions of the task specification by the TL. This could occur because the TL periodically updates how the task is specified given slight changes in her/his model of the mission. The UAS receive the task specification across different replan cycles. As a result, the UAS do not have sufficient common knowledge of the task to form consensus on which one should be assigned. (Similar to S-37.1.3)

- UAS have different beliefs about how many and which UAS are participating on the team. This may occur due to dynamic membership when UAS are periodically taken offline and added, and dynamic connectivity that prevents timely distribution of this information. As a result, UAS are unable to effectively coordinate on which one is best suited for the task. Alternatively, they could also come to consensus that a UAS that is not actually part of the team is best suited, and "relinquish" the search command to it, even though it will not do it. (Similar to S-37.1.3)
- UAS have different beliefs about the current roles and responsibilities of other controllers on the team. For example, one UAS believes that the TL has taken on the search task based on stale information, which gets disseminated to the rest of the team. This forces the control algorithms to not coordinate on which UAS should take it. (Similar to S-37.1.3)

- ***Model Updates*:** Elements of the UAS's process models are not adequately or consistently updated. ***Refinement***:
  - *Vertical Coordination (Control)*: The TL unknowingly over-specifies the search task, and imposes constraints that are too strict for the UAS team to reach an assignment solution. For example, TL specifies an execution time window that is unnecessarily narrow and cannot be satisfied by any UAS.[46]
  - *Lateral Coordination (Communication)*:
    - Communications between UAS are not adequate to execute a distributed decision-making algorithm, such as allocation of the search task. Too many coordination messages needed for the UAS to reach consensus are dropped, and the team does not converge on a solution. Communication channels may be degraded due to jamming, fading, and equipment damage. (Similar to S-37.1.3)
    - The information UAS use to run the distributed decision-making algorithm is inconsistent from UAS to UAS. For example, UAS1 receives a state estimate from UAS2, which becomes slightly outdated due to small communications and processing delays. Then for planning, it compares this now slightly outdated UAS2 state estimate with its own current UAS1 state estimate. The same effect also occurs when UAS2 plans. As a result, UAS plan on inconsistent information, and both conclude that the other UAS is better suited for the search task. (Similar to S-37.1.3)
  - *Lateral Coordination (Observation)*: (Similar to S-37.1.3)
  - *Prediction*: The UAS team assigns the search task using predictions of each other's future states. However, those projections are inaccurate and lead to lack of consensus or reaching an unfeasible solution. For example, UAS1 is tasked to fix at a certain time window. The location of that fix is used to predict where UAS1 will be and to determine that it will best be suited to

[46] (B) Scenario: task exceeds a limit for hydraulic system [or any other control limit], so does not actuate

execute the search. However, the fix task is later modified such that UAS1 can no longer meet the constraints of the search task. But by this time, it is too late to retask another UAS.[47]

- ***Decision-Making*:** The process UAS use to decide what control and communications actions they provide is inadequate or inconsistent across the team. ***Refinement***:
  - Despite adequate information sharing between UAS, the control algorithm is too slow to converge on a solution and "times out" without determining which UAS is most suitable to execute the task. (Similar to S-37.1.3)
  - Despite adequate information sharing between UAS, the control algorithm "churns". It replans or reoptimizes too frequently, overrides previous solutions before they can be executed, and ultimately the search task does not get performed. For example, UAS1 may be closer to the search task and deemed better suited to take it on. However, by the time this decision is made, UAS2, which was traveling for other reasons, is now slightly closer. The algorithm runs again, and re-assigns the task to UAS2 instead, but by then UAS1 has overtaken UAS2. The process repeats itself and no UAS performs the search. (Similar to S-37.1.3)
  - The UAS have a relatively consistent set of common information. But they each employ a non-deterministic decision-making algorithm (e.g., Machine Learning based) and reach inconsistent solutions for which UAS gets assigned the task. As a result, no UAS is assigned. (Similar to S-37.1.3)
  - The UAS have common planning information and are configured with common deterministic decision-making algorithms. However, their differences in contextual factors lead them to use the algorithm differently. For example, UAS1 has an internal system malfunction or is targeted by an enemy, and prioritizes resolving those issues over collaborating with UAS2. UAS2 does not receive coordination messages as expected. As a result, no UAS is assigned to search. (Similar to S-37.1.3)
- ***Capacity***: Same as S-37.1.3

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Changes in the set of UAS that participate in the team contribute to this UCCA. ***Refinement***:
  - The assigned UAS is then taken out of the team by a different controller (e.g., the Ground Station or a different team lead), or has an internal system failure that prevents it from executing its collaborative responsibilities. (Similar to S-37.1.3)
  - One of the UAS has an internal problem that leads it to repeatedly change control modes and its participation with the team. Every time the set of participants changes, the algorithm replans. This causes "churn" and no UAS performs the search task. (Related to Cognitive Alignment and Decision-Making above).
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***

[47] (B) Autonomous Controller calculates that the current state is sufficient to accomplish the task

- UAS2 is temporarily indirectly connected to the TL via UAS1. A new tasking provided by the TL and relayed to UAS2 by UAS1 conflicts with a previous tasking received by UAS2. Because of the source, UAS2 dismisses the new tasking and prevents the team from adequately coordinating a task assignment.

S-43.7.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. *The context of this UCCA does not apply to this top-level scenario.* (Same as S-37.1.4)

S-43.7.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL does not search (and does not task the UAS team to search). Emphasis is placed both on when the TL does not intend to search, and when s/he does intend to search but does not do so.[48] ***Refinement***:

- ***(Step 2: Internal Control) Unsafe Control Input, Unsafe Control Path:*** Similar to S-37.1.1
- ***(Step 2: Internal Control) Inadequate Process Model:*** Similar to S-37.1.1. In addition:
  - *Feedback not available.* TL is fixated on tasking UAS and forgets that s/he can search her/himself. Feedback from the interface highlights the UAS resources available, but it excludes resources the TL can contribute to the mission. (Similar to S-40.4.5)
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the need to search for a target and the team capabilities, but still chooses not to search or task a UAS to search. ***Refinement***:
  - TL is currently busy and prioritizes other tasks, but intends to search or task a UAS. However, s/he then forgets due to heavy workload. (Similar to S-37.1.1)
  - TL misunderstands the system and incorrectly believes that the UAS that are not tasked to fix or fire will default to executing preplanned searches. This was how the system was configured in past training exercises.[49]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Does not apply.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
  - ***Model Update*:**
    - *Prediction:* TL intends to search. However, s/he predicts that one of the UAS will navigate across the search area and should by default have its search sensor pointed out to gather information. The TL predicts the intent of the search task will be fulfilled by the UAS without formal tasking. However, the UAS takes a different path or does not collect data as anticipated. (Relates to construction of control algorithm).
  - ***Capacity*:** TL intends to perform the search task but is too busy monitoring the activities of the other UAS.[50]

---

[48] (B) UCA 25: The Team Lead does not provide waypoints for his FMS (H2).

[49] (B) Scenario: Team Lead believes the AuC can track targets [or search] without providing an explicit command, so TL does not provide track target [or search] command.

[50] (B) Scenario: Team Lead is so focused on the UAV(s) locations that he forgets to update his own position.

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Similar to S-37.1.1
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
    - TL starts to execute the search task but, but then anticipates that performing this will disconnect her/him from the team. S/he intends to retask another UAS to do it, but does not do so due to factors described in S-37.1.1.
    - The UAS team is tasked to search, but some of the UAS are indirectly connected to the TL, and receive their commands by relay through other UAS. Because the relays lack authority, those receiving the tasking indirectly reject it as invalid and do not perform it if selected by the automation. (Similar to S-37.1.3)

S-43.7.9 (Step 4: Other Factors): No member of the team searches due to factors beyond those explored in interactions between the controllers on the team. [DA]

- *Inadequate Process Feedback:* the controller performing the search receives inadequate state estimates to control its trajectory. This could be due to a system failure or interference.
- *Unsafe Process Behavior:* the TL or UAS does provide a search, but the target outmaneuvers the search process or conceals itself to make it ineffective.

UCCA 44.8 (abstracted UCCA): Controller Ci provides search and no other Cj provides search when the tasked entity is the only one capable for a higher-priority task and teammate can search [H3] [DA]

> **UCCA 44.8.1 (UCCA):** TL searches, UAS1 does not search, and UAS2 does not search when TL is the only one capable for higher-priority task and any UAS can search.
>
> **UCCA 44.8.2 (UCCA):** TL does not search, UAS1 searches, and UAS2 does not search when UAS1 is the only one capable for higher-priority task and TL or UAS2 can search.

The figure below shows how internal control combinations relevant to the UCCA and traces them to top-level scenarios. Case 1b is not analyzed for similar reasons as in previous UCCAs.

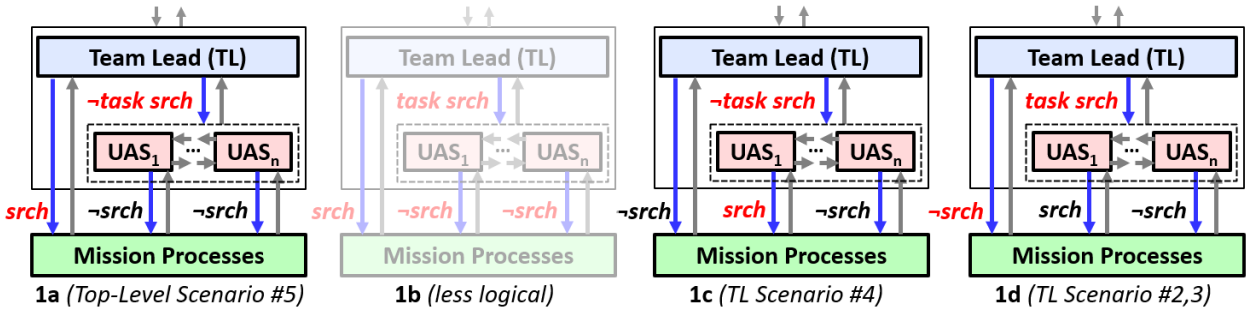


Figure A3-12. Internal control combinations that result in the wrong UAS providing the search

S-44.8.1 (Step 1: Top-Level Scenarios #1): TL does not direct the UAS as necessary for the team to execute safe collective control. *The context of this UCCA does not apply to this top-level scenario.* (Similar to S-38.2.1)

S-44.8.2 (Step 1: Top-Level Scenarios #2): TL directs the UAS in a way that leads to unsafe collective control. Here, TL tasks the UAS team to search, even though the selected UAS is needed for a higher-priority task, instead of executing the search task her/himself. [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** (Same as S-38.2.2)
- ***(Step 2: Internal Control) Inadequate Process Model:*** (Same as S-38.2.2, S-41.5.1). **In addition**:
    - TL does not have sufficient working knowledge of the multi-UAS control algorithm to anticipate which UAS will be selected to search if the team is tasked. This may be due to lack of transparency or feedback in the distributed task allocation algorithm. The TL incorrectly believes that the automation will not assign the UAS that is needed for a higher-priority task, or that it will update the assignment later as needed. (Same as S-44.8.5)
    - TL over-trusts the automation to produce an effective task assignment. TL over-relies on the automation to assign UAS in a way that is consistent with task priorities, even though it will not.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:***
    - TL does not adequately prioritize tasks to perform. S/he selects the search as the most important task to provide to the UAS team, and instead of prioritizing something else. (Similar to S-44.8.5)
- ***(Step 2: Internal Control) Unsafe Control Path:*** (Same as S-37.1.1). **In addition**:
    - TL intends to task the UAS onto the higher-priority task, but s/he unintentionally tasks the team to search. (Can be due to inadequate interface as in S-37.1.1)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*****:** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*****:** *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - TL anticipated that the UAS that would be assigned would not be the one needed for a higher-priority task. However, the UAS expected to be selected is taken offline by another controller (e.g., Ground Station or different team lead). As a result, the algorithm selects a UAS different than the one the TL expected.
    - TL assigns the search task to the UAS team, knowing that it will likely be assigned to a UAS that is needed for a higher-priority task. However, TL also anticipates that a new UAS will be added to the team shortly and that the algorithm will reassign the search task to the UAS at that time. This does not occur (e.g., the new UAS is unable to join the team, the algorithm does not replan when it joins, …).
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** No scenario conceived.

S-44.8.3 (Step 1: Top-Level Scenarios #3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, the UAS team is tasked, but the wrong UAS conducts the search. The only UAS capable of performing a higher-priority task is assigned, and other UAS that can conduct the search are not.[51] [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:***

[51] (B) UCA 30: Autonomous Controller implements a task using the wrong UAV (H3)

- The UAS team is overridden by another controller (e.g., the Ground Station, another team lead, or a cyber attacker) that designates the wrong UAS to conduct the search. (Similar to S-37.1.3)

- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.3
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:***
  - *Feedback about Controlled Process from Collaborators*: UAS2 (assigned to search) shares its feedback with the team. UAS1 uses this feedback to update its model of the search task (e.g., larger area to survey, needs increased precision), and determines that it is the only UAS that can execute the search. The team reassigns UAS1 to search even though it is need for a higher-priority task. (Similar to S-43.7.3)
  - *Feedback about Collaborator Control Actions from Controlled Process*: N/A
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The UAS on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
  - ***Construction***: The process models and/or control algorithms of the UAS are not adequately or consistently built across the team to support collaborative control. ***Refinement***:
    - The control algorithms on the UAS are not consistent with how the TL specifies taskings. Specifically, UAS do not account for task prioritization in the same way that the TL does. For example, TL simultaneously tasks the UAS team in general to conduct a search and UAS1 specifically to provide a <fix, fire>. The automation selects UAS1 for the search task and prioritizes the search over the other TL-directed task.[52]
    - The control algorithms on the UAS are not consistent with one another. Specifically, some UAS(s) are configured to treat a search task as an individually directed task from the TL (similar to fix, fire tasks), and not as a team task that gets automatically allocated. In this example, UAS1 is configured as such and interprets the search task as an individual tasking. It takes it on regardless of how the rest of the UAS team wants to allocate it, and regardless of TL intent.
  - ***Initialization***: Elements of the UAS's process models are not adequately or consistently initialized across the team. ***Refinement***:
    - UAS receive different versions of the task specification from the TL. This occurs because the TL periodically updates how the task is specified given slight changes in her/his model of the mission. The UAS receive the task specification across different replan cycles (Similar to S-43.7.3) and reach consensus over initial conditions that diverge from the intent of the TL.

[52] (B) Scenario: AuC algorithm calculates that other tasks within its system have a higher-priority. algorithm determines that the task is not a priority, so Auc does not implement command.

        - For example, TL initially specifies a high priority for the search task. A higher-priority <fix, fire> arises and the TL updates the initial tasking accordingly. However, multiple UAS do not receive the updated parameters, still operate on the belief the search is high priority, and outvote UAS(s) that received the updated parameter.
    - UAS have different beliefs about the current roles and responsibilities of other controllers on the team. For example, certain UAS(s) believe that the TL or another UAS has taken on the higher-priority task based on stale information, and they disseminate this information across the team. This causes the UAS team to then incorrectly assign UAS1 to execute the search, when in fact it is needed for that higher-priority task. (Similar to S-43.7.3)
- ***Model Updates*:** Elements of the UAS's process models are not adequately or consistently updated. ***Refinement***:
    - *Vertical Coordination (Control)*:
        - TL imposes different tasking constraints on different UAS that makes some of them unavailable to do the search. As a result, the UAS team has no other option but to allocate the search to the UAS that is needed for a higher-priority task.
        - TL tasks search in such a way that the UAS team misinterprets it as a command to task a specific UAS. However, that UAS is the only one that can accomplish the other higher-priority task.
    - *Lateral Coordination (Communication)*:
        - UAS share and use inconsistent state information in selecting the optimal UAS to execute the search. This can be due to inadequate communication channels between them, or delays in processing and sending information. UAS1 incorrectly determines that it is the only UAS that can execute the search. Thus, UAS1 searches instead of doing the higher-priority task. (Similar to S-37.1.3)
        - UAS share parameters about different tasks without specifying which one they are describing. Each teammate incorrectly correlates the information received to the wrong task. For example, UAS1 shares that it is the only UAS capable of executing a different task, but the UAS team interprets this statement as UAS1 being the only UAS capable of executing the search. (Similar to S-37.1.3)
    - *Lateral Coordination (Observation)*: UAS1, which was not selected to search, observes the UAS that was selected to search (e.g., UAS2) not maneuvering in a way consistent with that task. As a result, UAS1 loses confidence that UAS2 will execute the search, and is programmed to take over that task in such cases. Thus, UAS1 takes on the search and does not execute the higher-priority task. (Similar to S-43.7.3)
    - *Prediction*: *No scenario conceived.*

    - ***Decision-Making*:** The process UAS use to decide what control and communications actions they provide is inadequate or inconsistent across the team. ***Refinement***:
        - The control algorithm does not converge on a solution and "times out" without reaching consensus on which UAS is most suitable to execute the task. (See "Cognitive Alignment" in S-43.7.3). The UAS team is programmed with an arbitration heuristic that forces the assignment to UAS1 (e.g., assigns lowest ID number). UAS1 takes on the search without considering other tasks it is assigned, including the higher-priority tasks.
    - ***Capacity*:** The capacity of one of the controllers is inadequate to enable effective alignment of team cognition. (Same as S-43.7.3 and "Decision-Making" above)
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Changes in the set of UAS that participate in the team contribute to this UCCA. ***Refinement***:
    - The team of UAS adequately allocates the search to a UAS (e.g., UAS2) that does not have higher-priority tasks, unlike UAS1. However, UAS2 is then taken out of the team by a different controller (e.g., the Ground Station or a different team lead), or it has an internal systems failure that prevents it from searching. (Same as S-43.7.3)
    - The automation is programmed to retask UAS2's responsibilities to UAS1 if UAS2 drops out. UAS1 is now tasked to search, and drops its higher-priority task.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** TL had previously tasked the UAS team to search. A higher-priority task then arises. The TL intends to retask the UAS executing the search with these higher-priority tasks. However, the UAS off conducting the search is now disconnected and cannot be retasked. (Similar to S-43.7.3)

S-44.8.4 (Step 1: Top-Level Scenarios #4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, TL intends to search but instead, an untasked UAS better suited for a higher-priority task executes the search.[53] [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-44.8.3
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** *No scenario conceived.*
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Does not apply.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
    - **Construction**: The process models and/or control algorithms are not adequately or consistently built across the team. ***Refinement***:
        - TL has created the search task but has not yet assigned it to the UAS. TL assumes, based on working knowledge of the autonomation, that the UAS

[53] (B) UCA 31: Autonomous Controller implements a task when there is no command from an authorized command provider (H3)

will not take on this task. However, one or more of the UAS is programmed to execute search tasks in the queue by default, if not tasked with anything else. This unexpected UAS search behavior confuses the TL. S/he believes the UAS may have been actively tasked by another controller (e.g., Ground Station or different team lead), does not retask the UAS to the higher-priority task, and does not execute the search her/himself.

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** *No scenario conceived*
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** TL had previously tasked the UAS team to search. A higher-priority task then arises. The TL intends to relieve the UAS from executing the search and do the search her/himself. However, the UAS off conducting the search is now disconnected and cannot be untasked.[54]

S-44.8.5 (Step 1: Top-Level Scenarios #5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL searches, even though s/he is needed for a higher-priority task, instead of tasking the UAS team to search. [DA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** (Same as S-38.2.2)
- ***(Step 2: Internal Control) Inadequate Process Model:*** (Same as S-38.2.2, S-41.5.1). Additional related factors:
  - TL does not have sufficient working knowledge of the multi-UAS control algorithm to anticipate which one will be selected to search if the team is tasked. This may be due to lack of transparency or feedback in the distributed task allocation algorithm. TL intends to task a specific UAS for a later task and decides not to risk tasking the team to search so that the one UAS is available (related to Inadequate Control Algorithm).
  - TL does not have confidence the automation can assign tasks effectively. Instead, TL decides to not rely on it and performs all tasks her/himself.[55]
- ***(Step 2: Internal Control) Inadequate Control Algorithm:***
  - TL mis-prioritizes the search as the most important command and then elects to do it her/himself, instead of focusing on a more important task.
  - TL believes s/he can execute the search task while enroute to the higher-priority task (e.g., multi-task). However, s/he ends up more consumed by the search task than anticipated and is unable to execute the other command.
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task the UAS team to search, but cannot for the following reasons. As a result, the TL searches her/himself.
  - TL gets lost in the user interface and is not capable of tasking the search.
  - The control communication channel is inadequate (e.g., jamming, link failures).
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:***
  - *Feedback about Controlled Process from Collaborators*: A UAS is assigned to search, and shares its feedback with the team. The TL receives degraded information

[54] (B) Scenario: There is a delay in another command from the Team Lead to delete the command. There is a transmission error, delay, communication link failure, so the AuC does not receive the command.
[55] (B) Scenario: Team Lead identifies other targets based on only his visual because the AuC provided false targets earlier in the mission.

regarding the search (e.g., degraded communications) and decides to intervene by taking over the search, even though it was not necessary.
    - *Feedback about Collaborator Control Actions from Controlled Process*: N/A
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The UAS on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
    - ***Construction***: Same as S-41.5.5
    - ***Initialization***: Same as S-41.5.5, **in addition**:
        - The controllers have different beliefs about the set of controllers on the team. The UAS team allocates the search task to a UAS that the TL does not believe is a part of the team, or believes is incapable of searching. The TL intervenes and carries out the search. (Similar to S-43.7.3)
    - ***Model Updates***: Elements of the UAS's process models are not adequately or consistently updated. ***Refinement***:
        - *Vertical Coordination (Control)*: TL continuously adjusts inputs to try to get the UAS team to select a specific UAS to execute the search, or to get the UAS to execute the search in a certain way. However, the automation never outputs the expected/desired solution. So, the TL cancels the tasking and executes the search her/himself.
        - *Lateral Coordination (Communication)*: UAS have inconsistent state or planning information that leads to a lack of consensus or a suboptimal solution on how to perform the search task. Either are unsatisfactory to the TL, who cancels the task and does it her/himself.
        - *Lateral Coordination (Observation)*: Same as S-41.5.5
        - *Prediction*: Same as S-41.5.5
    - ***Decision-Making***: See all the model update scenarios above.
    - ***Capacity***: Same as S-40.4.5
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-41.5.5
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S-44.8.9 (Step 4: Other Factors): The "wrong" controller provides the search due to factors beyond those explored in interactions between the controllers on the team. [DA]

- No new scenarios conceived.

Abstraction 2b – Type 3-4: Controllers start / end providing control actions too early / too late relative to one another

The following are the causal scenarios associated with all Abstraction 2b – Type 3-4 UCCA. As in the analysis above, each high-level UCCA is analyzed together with its refined UCCAs. Scenario identification for each UCCA starts with the following three top-level scenarios for all Type 3-4 UCCAs. These are intended to provide focus and coverage over different possible control actions internal to the team that could lead to unsafe collective output. The fourth top-level scenario covers other factors in the same way as for Type 1-2 UCCAs.

6. Directed Sequence Unsafe: TL directs the UAS in a way that leads to unsafe temporal sequencing.

7. Directed (Safe) but Executed in Unsafe Sequencing: TL adequately directs the UAS, but the way in which the UAS execute the tasks leads to unsafe temporal sequencing.
8. Controller Actions to Process and Directions Unsafe in Sequencing: TL control actions to the shared process are unsafe in temporal sequencing with how it directs the UAS.
9. Other: factors beyond those explored in interactions between the controllers on the team.

UCCA 47.1 (abstracted UCCA): Controller Cj starts providing fix command before Ci ends providing fix command when that creates mutual interference [H1, H3] [TA]

UCCA 47.1.1 (refined): UAS2 starts providing fix before UAS1 ends providing fix.

The figure below shows the internal control actions relevant to the UCCA and traces them to the top-level scenarios. S(fix) and **F** E(fix) is read as "starts fix before ends fix", where ***F*** is the Linear Temporal Logic (LTL) operator for *some **F**uture step*.

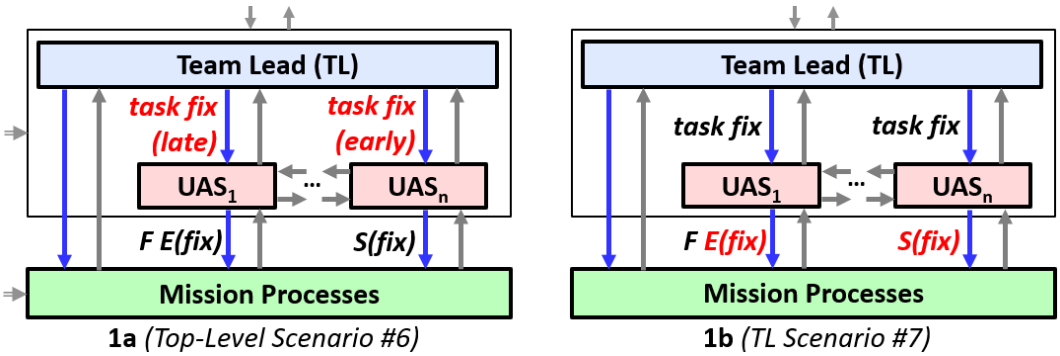


Figure A3-13. Internal control actions that result in mutual interference in fix task handoff

S-47.1.6 (Step 1: Top-Level Scenarios #6): TL directs the UAS in a way that leads to unsafe temporal sequencing. Here, TL tasks a UAS to start fix too soon and another to end fix too late relative to each other. [TA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL misinterprets direction from higher authorities to be concerned about <gaps, overlaps> between controllers that are handing off a <fix, search>.[56]
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: (1) the time windows tasked do not constitute excessive <gap, overlap>, (2) UAS(s) will not actually provide the fix over their entire time window, (3) multiple UAS will not cause hazardous effects if they fix simultaneously.[57] ***Refinement***:
    - *Missing Feedback.* TL interface does not provide a way to effectively compare UAS tasking timelines.

[56] (B) Scenario: [higher authorities] dis-authorization comes across as an authorization due to inaccuracies in the radio.

[57] (B) Scenario: TL has accurate feedback related to timing, but has an inaccurate mental model of how long it takes the Automated Controller to implement a command upon receiving it, so TL issues command late.

- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the target and the timing associated with controllers on the team, but still chooses to task a UAS to <fix, search> in a way that creates a control <gap, overlap>. ***Refinement***:
    - TL is busy and prioritizes other operating tasks. S/he provides an initial tasking to get the team moving, but plans to update the tasking of the 2nd UAS fixing before the unsafe handoff. However, s/he later forgets due to heavy workload.[58]
    - TL misunderstands how the UAS operate. S/he believes that the UAS will coordinate with one another to avoid the unsafe handoff.
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task the UAS <deconflicted, adjoining> time windows, but is unable to do so. ***Refinement***:
    - The communication channel to one or more of the UAS is inadequate (e.g., RF jamming, communications fading, misconfigured encryption settings, …).
        - This prevents TL from updating UAS1's tasking after it has specified UAS2's time window.
        - This prevents a UAS from receiving the full task information (e.g., UAS receives the task and start time, but not the end time) and turns to preprogrammed default parameters to fill in the information gap.[59]
    - The interface has a complicated or confusing workflow to specify a task. This is aggravated in high workload environments. For example, tasks are programmed with default durations based on specified start times. Those durations are not automatically adjusted based on the timelines associated with other controllers.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*:** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*:** *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - The set of UAS participating fluctuates so much that the TL has difficulty effectively managing their contributions. For example, TL had previously tasked UAS to perform a proper handoff. However, some of the UAS are taken offline by a different controller (e.g., Ground Station or different team lead). The TL rushes to recreate the tasking and makes a mistake that leads to the <gap, overlap>.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** No new scenarios conceived

S-47.1.7 (Step 1: Top-Level Scenarios #7): TL adequately directs the UAS, but the way in which the UAS execute the tasks leads to unsafe temporal sequencing. Here, UAS2 starts to provide the fix before UAS1 ends it. [TA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Another controller (e.g., Ground Station, another team lead, cyber attacker) directly overrides one of the UAS (e.g., directs it to

[58] (B) Scenario: Team Lead is pre-occupied with another task, so provides an identification command late.
[59] (B) Scenario: AuC uses an old algorithm that defaults to a specific payload, so does not provide a command that powers on the appropriate payload.

start/end providing the fix command) in a way that conflicts with the team coordinated task handoff process. (Similar to S-37.1.3)

- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** UAS1 intends to hand off the fix command to UAS2 at the coordinated time. However, its targeting equipment malfunctions and it is unable to turn it off in time. Similarly, UAS2 intends to start at the coordinated time, but its equipment malfunctions and it unintentionally starts providing the fix early.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** A conceivable mechanism multiple controllers can use to coordinate a control handoff is to use feedback from the shared process to mutually estimate each other's actions. In this case, Cj can detect a change in fix feedback from the process, correlate it with Ci ending its fix command, and use that information to determine when to start its fix command. Ci and Cj also receive feedback on their actions from other controllers on the team. For example, Ck (tasked to fire) provides feedback on the fix, as described in S-37.1.3. In such cases, controllers mutually close each other's control loops (Figure A3-14). ***Refinement***:

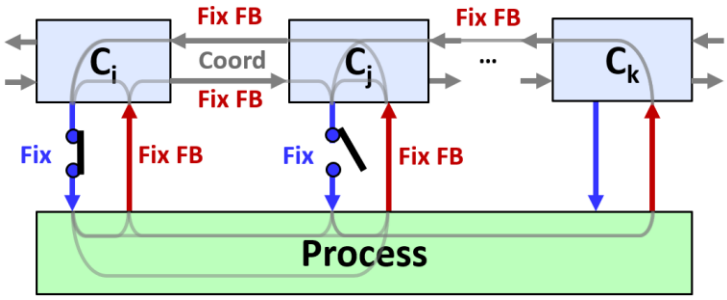


Figure A3-14. Control structure for controllers collaborating on handing off a fix

  - ***Feedback about Controlled Process from Collaborators*:**
    - The feedback UAS1 receives from another UAS leads it to believe UAS2 has not started the fix and that UAS1 should continue to provide it. ***Refinement***:
      - UAS1 ends its fix in anticipation of UAS2 starting it. A teammate then provides feedback that the fix is not (or is no longer) adequate to support a fire command for an excessive period of time. This may be due to other factors beyond UAS2 not providing a fix. However, UAS1 interprets this as UAS2 not taking over control, and resumes its own fix, which now interferes with UAS2.
      - UAS1 ends its fix as planned. UAS2 provides feedback (correctly) to UAS1 that it no longer sees its fix. However, UAS1 misinterprets this as a request to continue providing a fix. As such, it resumes its own fix, which now interferes with UAS2.
    - The feedback UAS2 receives from another UAS leads it to believe UAS1 has ended the fix and that UAS2 should start providing it. ***Refinement***:

- UAS1 has not yet ended its fix (perhaps as planned). However, a teammate provides feedback that the fix is not (or is no longer) adequate to support a fire command. This may be due to other factors beyond UAS1 not providing a fix. However, UAS2 interprets this as UAS1 relinquishing control, and begins its own fix, which now interferes with UAS1.

  - ***Feedback about Collaborator Control Actions from Controlled Process*:**
    - The feedback UAS1 receives from the process leads it to believe it UAS2 has not started the fix and that UAS1 should continue providing it. ***Refinement***:
      - UAS1 ends its fix in anticipation of UAS2 starting. But it then does not receive fix feedback from the process consistent with UAS2 now providing a fix. This may be due to other factors beyond UAS2 not providing a fix (e.g., sensor malfunction, misconfiguration, …). However, UAS1 interprets this as UAS2 not taking over control, and resumes its own fix, which now interferes with UAS2.
    - The feedback UAS2 receives from the process leads it to believe UAS1 has already ended the fix and that UAS2 should start providing it. ***Refinement***:
      - UAS1 has not yet ended its fix (perhaps as planned). However, UAS2 does not receive fix feedback from the process consistent with UAS1 still providing a fix. This may be due to factors beyond UAS1 not providing a fix (e.g., misconfiguration, sensor failure). However, UAS2 interprets this as UAS1 relinquishing control, and begins its own fix, which now interferes with UAS1.

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement:***
  - ***Construction***: The process models and/or control algorithms are not adequately or consistently built across the team to support collaborative control. Refined:
    - (Human-Machine) The control algorithms on the UAS are not compatible with how the TL specifies the <fix, search> task. This prevents the controller from accepting the task or coordinating specific time windows for its handoff. This could occur due to a misconfiguration of the UAS or TL interface software, variation in how the TL was trained to work with the UAS, and conflicting past experience the TL has in working with human wingmen instead. (Similar to S-37.1.3)
    - The control algorithms on the different UAS are not compatible with one another. As a result, the UAS lack the ability to coordinate a fix command handoff. This could be due to a configuration management issue. (Similar to S-37.1.3)
  - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
    - The different UAS receive different versions of the fix task specification by the TL. This occurs because the TL periodically updates how the task is

specified given slight changes in her/his model of the mission. The UAS receive the task specification across different replan cycles. As a result, the controllers do not have sufficient common knowledge of the task to coordinate its handoffs. (Similar to S-37.1.3)

- The different UAS have different beliefs about how many and which UAS are participating on the team. This occurs dues to dynamic membership, with UAS cycling on and offline, and dynamic connectivity that prevents timely distribution of this information. This could result in the UAS1 not believing that UAS2 is currently participating in the team. As such, UAS1 does not believe UAS2 is capable of accepting a handoff and continues to provide its fix beyond its time window. (Similar to S-37.1.3)
- UAS have different beliefs about the current roles and responsibilities of other controllers on the team. For example, UAS1 is not informed about which teammate is responsible for taking over <fix, search> task after its time window. As such, it does not know who to coordinate the handoff with. (Similar to S-37.1.3)

- ***Model Updates***: Some elements of some of the process models are not adequately or consistently updated across the team. ***Refinement***:
  - *Vertical Coordination (Control)*:
    - TL controls coordination of the handoff between UAS1 and UAS2 (i.e., TL commands UAS1 to end and UAS2 to start the command). This can be unsafe for reasons listed in S-47.1.6.
    - TL overrides one of the UAS with a command inconsistent with the laterally coordinated task handoff. For example, UAS1 and UAS2 have laterally coordinated how to do the handoff and are on track to complete it. However, TL accelerates the UAS2 timeline to make it available for other tasks, but s/he is not aware that this now conflicts with the coordinated plan.
  - *Lateral Coordination (Communication):*
    - Communications channels between the two controllers assigned the <fix, search> command are not adequate to coordinate on the task handoff. Too many coordination messages needed to reach consensus on when or how to provide the handoff are dropped. Communication channels may be degraded due to jamming, fading, and equipment damage. (Similar to S-37.1.3)
    - The information controllers use to coordinate the handoff is inconsistent. For example, UAS1 receives a state estimate from UAS2, which becomes slightly outdated due to small communications and processing delays. It uses this slightly outdated UAS2 state estimate and incorrectly concludes that UAS2 is not yet in position to handoff the fix command, and UAS1 continues to provide a fix. Conversely, UAS2 uses its up-to-date

state estimate to determine it is in position for the handoff, and starts its command. (Similar to S-37.1.3)

- A UAS is temporarily disconnected from the team and its model of the task handoff from that of the team. The UAS reconnects and now contributes its divergent variables. That disturbs team consensus regarding how to execute the handoff. (Same as S-37.1.3)

- *Lateral Coordination (Observation)*: UAS1 observes another UAS2 maneuvering in a way that is inconsistent with accepting a handoff of the fix command. It therefore incorrectly decides to keep providing its fix command too late. Similarly, UAS2 observes UAS1 maneuvering in a way that is consistent with handing off the fix command and incorrectly starts providing its own fix too early.
- *Prediction*: UAS1 has a model of the fix command handoff process that expects to receive certain coordination messages and observations by certain milestones. If UAS2 is delayed or takes an unexpected trajectory to fire, UAS1 may lose confidence in its ability to hand off the task and could decide not to <end, start> its fix command. (Similar to S-37.1.3)

- ***Decision-Making***: The process controllers use to decide what control and communications actions they provide are inadequate or inconsistent across the team. ***Refinement***:
  - The dynamics of the handoff process and the associated distributed decision-making lead to churn. For example, UAS1 ends its fix task to hand it off to UAS2. After a delay, UAS2 starts its fix task to complete the handoff. However, that delay was enough for UAS1 to assume no handoff took place, and it therefore it resumes its fix command. This leads to UAS2 detecting a conflict and stopping its task before UAS1 has a chance to do so. This handoff cycle repeats, and alternates between the UAS interfering with each other or leaving a gap in the fix.
  - Automated controllers employ a non-deterministic decision-making algorithm (e.g., Machine Learning based) that leads to unpredictable or inconsistent control and information outputs. This interferes with reaching consensus over the task handoff process. (Same as S-37.1.3)
  - UAS1 has low confidence in the ability of UAS2 to execute the handoff. As a result, UAS1 precautionarily chooses not to end its fix. (Same as S-37.1.3)
- ***Capacity***: The capacity of one of the controllers is inadequate to enable effective alignment of team cognition. ***Refinement***: (Same as S-37.1.3)

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
  - The loss of a teammate makes UAS1 continue to provide the fix instead of handing it off. For example, UAS1 was previously paired with another UAS for the handoff of the fix. That UAS is taken offline (by another controller, lost in mission, …) and the handoff is reassigned to UAS2. However, the retasking is not known to UAS1, and therefore, it does not know to coordinate a handoff with UAS2.

- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived. Changes in network topology can contribute to the unsafe coordination factors listed above.*

S-47.1.8 (Step 1: Top-Level Scenarios #8): TL control actions to the shared process are unsafe in temporal sequencing with how it directs the UAS. *The context of this UCCA does not apply to this top-level scenario.*

Note: *The TL does not provide the fix control action.*

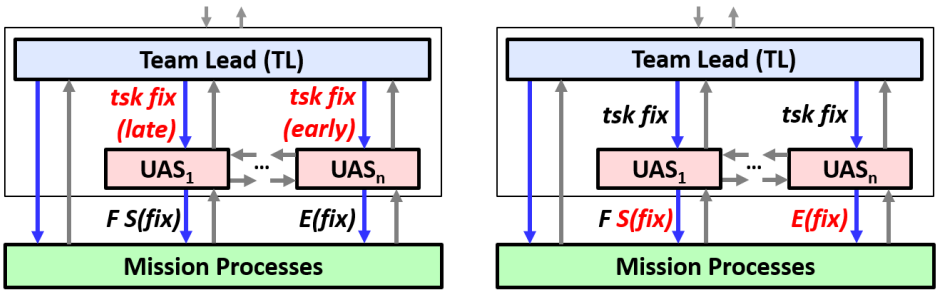
S-47.1.9 (Step 4: Other Factors): UAS2 starts to provide the fix before UAS1 ends it due to factors beyond those explored in interactions between the controllers on the team. [TA]

- ***Unsafe Process Behavior:*** UAS1 intends to hand off the fix command to UAS2 at the coordinated time, but the target deploys countermeasures or performs defensive maneuvers that overcome the fix effects and make UAS2 choose to <start, end> early.[60]

UCCA 48.2 (abstracted UCCA): Controller Cj ends providing fix command before Ci starts providing fix command when that creates a large gap in a fix handoff [H3] [TA]

UCCA 48.2.1 (refined): UAS2 ends providing fix before UAS1 starts providing fix.

The figure below shows the internal control actions relevant to the UCCA and traces them to the top-level scenarios. ***F*** is the LTL operator for *some **F**uture step*.



1a *(Top-Level Scenario #6)* **1b** *(TL Scenario #7)*

Figure A3-15. Internal control combinations that result in an excessive gap in fix task handoff

S-48.2.6 (Step 1: Top-Level Scenario 6): TL directs the UAS in a way that leads to unsafe temporal sequencing. Here, TL tasks a UAS to end fix to soon and another to start fix too late relative to each other. [TA]

- ***All factors:*** Same as S-47.1.6

S-48.2.7 (Step 1: Top-Level Scenario 7): TL adequately directs the UAS, but the way in which the UAS execute the tasks leads to unsafe temporal sequencing. Here, UAS2 ends the fix command before UAS1 starts. [TA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-47.1.7
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.

[60] (B) Scenario: An attacker provides a false fix on target command to the FMS

- ***(Step 2: Internal Control) Unsafe Control Path to Process:***
    - UAS1 intends to take on the fix command from UAS2 at the coordinated time. However, its targeting equipment malfunctions and it is unable to turn it on in time. Similarly, UAS2 intends to end at the coordinated, but its equipment malfunctions, and it unintentionally ends the fix early. (Similar to S-47.1.7)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same illustration as in S-47.1.7 in Figure A3-14. ***Refinement***:
    - ***Feedback about Controlled Process from Collaborators*:**
        - The feedback UAS1 receives from another UAS leads it to believe UAS2 has not ended the fix and that UAS1 should not start to provide it. ***Refinement***:
            - UAS2 ends its fix in anticipation of UAS1 starting it. A teammate then provides feedback that the fix is adequate to support a fire command. This may be due to other factors beyond UAS2 providing a fix. However, UAS1 interprets this as UAS2 not ending its control and does not start its own fix, which leads to a gap in execution. (Similar to S-47.1.7)
        - The feedback UAS2 receives from another UAS leads it to believe UAS1 has started the fix and that UAS2 should stop providing it. ***Refinement***:
            - UAS1 has not yet started its fix (perhaps as planned). However, a teammate provides feedback to UAS2 that its fix is not (or is no longer) adequate to support a fire command. While this may be due to other factors beyond UAS1 providing a fix, UAS2 interprets this as UAS1 providing a fix in addition to UAS2 and creating interference. As a result, UAS2 prematurely stops providing the fix and creates an excessive gap in handoff. (Similar to S-47.1.7)
    - ***Feedback about Collaborator Control Actions from Controlled Process*:**
        - The feedback UAS1 receives from the process leads it to believe it UAS2 has not ended the fix and that UAS1 should not start its fix. ***Refinement***:
            - UAS2 ends its fix in anticipation of UAS1 starting. But UAS1 then continues to receive fix feedback from the process consistent with UAS2 still providing a fix. This may be due to other factors beyond UAS2 providing a fix (e.g., stale feedback, another UAS providing an erroneous and spurious fix, …). However, UAS1 interprets this as UAS2 not relinquishing control, and waits to start providing the fix. This creates an excessive gap in execution. (Similar to S-47.1.7)
        - The feedback UAS2 receives from the process leads it to believe UAS1 has started the fix and that UAS2 should stop providing it. ***Refinement***:
            - UAS1 has not yet started its fix (perhaps as planned). However, UAS2 still receives fix feedback from the process consistent with UAS1 providing an interfering fix. This may be due to other factors including a sensor malfunction or a spurious fix from another

controller. UAS2 interprets this as UAS1 taking over control and its own fix, which creates an excessive gap. (Similar to S-47.1.7)

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement:***
  - ***Construction*:** (Same as S-47.1.7)
  - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
    - The different UAS receive different versions of the fix task specification by the TL. (Same as S-47.1.7)
    - The different UAS have different beliefs about the current roles and responsibilities of other controllers on the team. (Same as S-47.1.7)
  - ***Model Updates***: Some elements of some of the process models are not adequately or consistently updated across the team. ***Refinement***:
    - *Vertical Coordination (Control)*: (Same as S-47.1.7)
    - *Lateral Coordination (Communication)*
      - Communications between the two UAS assigned to fix are not adequate to coordinate the task handoff. (Same as S-47.1.7)
      - The information controllers use to coordinate the handoff is inconsistent. For example, UAS1 receives a state estimate from UAS2, which becomes slightly outdated due to small communications and processing delays. It uses this slightly outdated UAS2 state estimate and incorrectly concludes that UAS2 is still in position to provide the fix command and has not yet handed it off. So, UAS1 does not start to provide a fix. Conversely, UAS2 uses its up-to-date state estimate to determine it is ready to handoff and ends its command. (Similar to S-47.1.7)
      - A UAS, temporarily disconnected, reconnects and contributes its divergent model variables to the team. (Same as S-47.1.7)
    - *Lateral Coordination (Observation)*: UAS1 observes UAS2 maneuvering in a way that is inconsistent with handing-off the <fix, search> command. Thus, it incorrectly decides to not start providing its command until too late. Similarly, UAS2 observes UAS1 maneuvering in a way that is consistent with taking over the <fix, search> command, and incorrectly ends its own command too early. (Similar to S-47.1.7)
    - *Prediction*: (Same as S-47.1.7)
  - ***Decision-Making***: Same as S-47.1.7, but with this additional refined factor:
    - The distributed decision-making process is too slow to keep up with the dynamic state of the shared-controlled process. It is unable to converge on a solution for the handoff in time. (Similar to S-37.1.3).
    - The distributed decision-making process does not account for disturbances that affect the ability of UAS to fulfill their time windows commitments. For example, UAS1 is delayed due to a process disturbance (e.g., unplanned headwind), and is unable to make the time window for the

handoff. Similarly, UAS2 must drop out of the task early due to a process disturbance (e.g., unplanned excessive fuel burn), and is unable to make the time window for the handoff.[61]

  - ***Capacity***: (Same as S-47.1.7)
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** (Same as S-47.1.7)
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** (Same as S-47.1.7)

S-48.2.8 (Step 1: Top-Level Scenario 8): TL control actions to the shared process are unsafe in temporal sequencing with how it directs the UAS. *The context of this UCCA does not apply to this top-level scenario.* (Same as S-47.1.8)

S-48.2.9 (Step 4: Other Factors): A UAS ends its fix command before another UAS starts due to factors beyond those explored in interactions between the controllers on the team. [TA]

- Same as S-47.1.9

UCCA 56.3 (abstracted UCCA): Controller Cj ends providing search before Ci starts providing search when that creates an excessive gap in a search handoff [H3] [TA]

UCCA 56.3.1 (refined): TL ends providing search before UAS1 starts providing search.

UCCA 56.3.2 (refined): UAS1 ends providing search before TL starts providing search.

UCCA 56.3.3 (refined): UAS2 ends providing search before UAS1 starts providing search.

The figure below shows the internal control actions relevant to the UCCA and traces them to the top-level scenarios. ***F*** is the LTL operator for *some **F**uture step*. Items 1a-1b represent UCCA 56.3.1, items 2a-2b represent UCCA 56.3.2, and items 3a-3b represent UCCA 56.3.3.

[61] (B) Scenario: AuC algorithm miscalculates which UAV is optimal, algorithm does not account for environmental factors impacting the UAV's sensors.

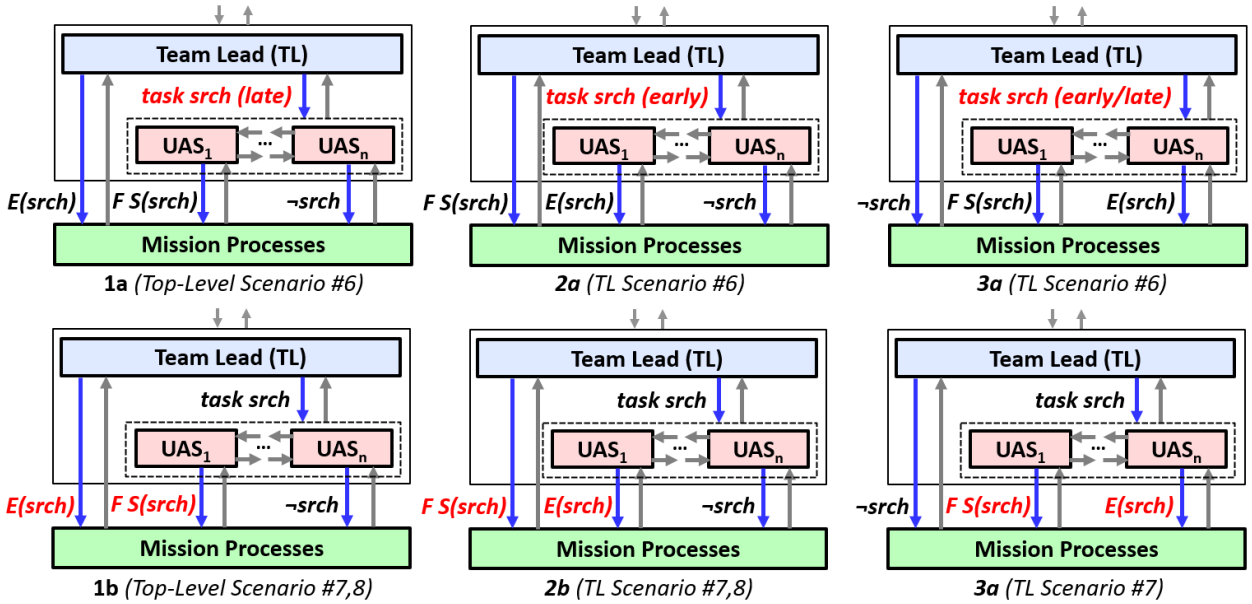


Figure A3-16. Internal control combinations that result in an excessive gap in fix task handoff

***Note**: the following scenarios assume the handoff of the search command is time-based (i.e., Ci searches over a time window then Cj searches over a later time window). The same reasoning can be applied to spatially based handoff is (i.e., Ci searches over one area then Cj searches over the remaining area).*

S-56.3.6 (Step 1: Top-Level Scenario 6): TL directs the UAS in a way that leads to unsafe temporal sequencing. Here, TL tasks the UAS to start or end the search in a way that creates a gap with the TL's planned start or end to the search. [TA]

> ***Note**: By system definition, the TL specifies a general search task for the UAS team, and the automation is responsible to assign UAS(s) to fulfill the task. This implies that the autonomy is responsible to manage the specification of handoffs between two UAS. This is analyzed in S-56.3.7.*

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-47.1.6
- ***(Step 2: Internal Control) Inadequate Process Model**:* Same as 56.3.8
- ***(Step 2: Internal Control) Inadequate Control Algorithm**:* TL has accurate information about the search requirements and the timing associated with controllers on the team. But s/he still chooses to perform the search and task the UAS team in a way that leads to an excessive search gap in handoff. ***Refinement***:
  - TL is busy and prioritizes other operating tasks. S/he provides the initial tasking as a placeholder for the team but intends to adjust the tasking or her/his execution before the interference. However, s/he later forgets to do this.[62]
- ***(Step 2: Internal Control) Unsafe Control Path**:* TL intends to task the UAS team time windows appropriate for a handoff to/from the TL, but is unable to do so. ***Refinement***:
  - TL makes an error in specifying the time (e.g., wrong format, invalid time,…) and the automation performs an auto-correct that is incompatible with the TL intent.

[62] (B) Scenario: Team Lead is pre-occupied with another task so provides command late

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops****: Not Applicable. The control loop to task the UAS is only closed through the TL.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment****: Not Applicable. This UCCA focuses on how the TL makes the decision to task the UAS.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - TL creates the search tasking knowing that a specific UAS is available in the time window for the handoff to/from the TL. However, that UAS is removed from the team (e.g., removed by another controller, shot down, …), and the UAS team is now unable to full fill the handoff timeline but does not inform the TL.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
    - TL creates the tasking by specifying how the UAS team should adapt the handoff in case of lost communications. Communications are then lost and the TL becomes unaware that its execution does not meet the parameters it specified for lost communications. The UAS drop the task and it creates a gap in the handoff (e.g., drop task too early, or drop plans to get handed off the task).

S-56.3.7 (Step 1: Top-Level Scenario 7): TL adequately directs the UAS, but the way in which the UAS execute the tasks leads to unsafe temporal sequencing. Here, UAS1 stops providing the search before UAS2 starts providing the search. [TA]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-47.1.7
- ***(Step 2: Internal Control) Inadequate Process Model****:* See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm****:* See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** UAS1 intends to hand off the search to UAS2 at the coordinated time. However, its search sensor equipment malfunctions and shuts down early. Similarly, UAS2 intends to start at the coordinated time, but its equipment malfunctions and it is late to turn on. (Similar to S-47.1.7)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops****:* Same as S-56.3.8, but between UAS1 and UAS2.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment****:* The UAS on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
    - ***Construction***: The process models and/or control algorithms are not adequately or consistently built across the team to support collaborative control. ***Refinement***:
        - The control algorithms on the different UAS are not compatible with one another. As a result, the UAS are not able to coordinate a safe handoff of the search command. (Similar to S-43.7.3)
    - ***Initialization***: Elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
        - The different UAS receive different versions of the task specification by the TL. This occurs because the TL periodically updates how the task is specified given slight changes in her/his model of the mission. The UAS receive the task specification across different replan cycles. While it can prevent the automation from selecting a UAS for the task, it can also select

a UAS that has a different model of the task and its handoff than what the other controller in the handoff <TL, UAS> expects. (Similar to S-43.7.3)
        - UAS have different beliefs about the current roles and responsibilities of other controllers on the team. (Same as S-47.1.7 applied to a search)
    - ***Model Updates***: Elements of the UAS's process models are not adequately or consistently updated. ***Refinement***:
        - *Vertical Coordination (Control)*: TL tasks a UAS involved in the search of another command that conflicts with the coordinated search handoff time.
        - *Lateral Coordination (Communication)*:
            - Communications between the two controllers are inadequate to coordinate the handoff. (Same as S-47.1.7)
            - Information provided by another UAS leads to a lack of consensus on the search handoff time. For example, a UAS shares stale information that one of the UAS involved in the search has another task that conflicts with the ability to meet the handoff time.
        - *Lateral Coordination (Observation)*: Same as S-48.2.7
        - *Prediction*: The UAS do not explicitly coordinate a handoff time. They use implicit coordination based on shared state information and common algorithms to anticipate when the task handoff will take place. However, there is an unanticipated disturbance in the state information shared or progress in execution, and the UAS do not update their predictions. The predictions are now invalid and lead to the excessive gap in handoff.
    - ***Decision-Making***: Same as S-47.1.7, but leads to a gap in the search handoff.
    - ***Capacity***: Same as S-37.1.3
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - A new UAS, which is able to take over the search early, joins the team. Given the change in team composition, UAS1 (tasked to execute the first part of the search) re-evaluates how the handoff is expected to occur. It assumes the new UAS will take over the search early, and as such, it stops searching before the coordinated handoff time. However, none of the other UAS, including the new one and the one originally slated to take over the search adjust to this new plan (e.g., due to inadequate coordination, and differences in decision-making), leading to the gap.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
    - The changing network topology leads to conflicting information between the two UAS coordinating the search handoff. For example, UAS1 initially sends a message that it will end the search at Time1. Because it is not directly connected to UAS2, that message is relayed through multiple hops to UAS2. Delays are introduced at every hop. Meanwhile, UAS2 becomes directly connected to UAS1 and they reach consensus that the handoff will occur at Time2. However, the original message regarding Time1 arrives at UAS2 after this direct exchange and is interpreted by UAS2 (only) as a change in the handoff time.

S-56.3.8 (Step 1: Top-Level Scenario 8): TL control actions to the shared process are unsafe in temporal sequencing with how it directs the UAS. Here, TL executes the search and tasks the UAS team to search in a way that leads to (1) the TL ending its search before a UAS starts to search, or (2) a UAS ending its search before the TL starts its search. [TA]

- ***(Step 2: Internal Control) Unsafe Control Input****:* (Same as S-47.1.6)
- ***(Step 2: Internal Control) Inadequate Process Model****:* TL has the following inadequate process model variables: (1) the time window the TL plans to execute the task, (2) the gap between the time window planned by the TL and that tasked to the UAS, (3) the ability of the UAS team to meet the tasked timeline, (4) the amount of time that is considered an excessive gap. ***Refinement***:
    - *Missing Feedback.* The interface does not provide a way to effectively compare the timelines associated with UAS taskings with the TL's own plans.
    - *Missing Feedback.* The interface does not show how close UAS are to not making the planned timeline (i.e., how sensitive the UAS are to unplanned disturbances).[63]
- ***(Step 2: Internal Control)*** *Inadequate Control Algorithm:* TL has accurate information about the search requirements and the timing associated with controllers on the team, but still chooses to perform the search and task the UAS team in a way that leads to an excessive search gap. ***Refinement***:
    - TL misunderstands how the UAS operate. S/he believes that the assigned UAS will observe the behavior of the TL and adjust its search window accordingly. For instance, the UAS will continue the search until the TL is ready to take over, or the UAS will start it early if the TL ends the first part of the search early.
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task the UAS team time windows appropriate for a handoff to/from the TL, but is unable to do so. ***Refinement*** same as S-47.1.6.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops****:* As described in S-43.7.3 (Figure A3-11), if the controllers share information gathered during the search to dynamically update their model of the remaining task, then this dynamic applies.
    - ***Feedback about Controlled Process from Collaborators***: The two controllers responsible to conduct the search via handoff are identified (TL and UAS1) and pre-coordinate a handoff time. The first controller <TL, UAS1> executes its part of the search, and during execution, gathers data that significantly changes the scope of the remaining task. However, this search feedback is not adequately received by the second controller <UAS1, TL>, or is received but cannot be acted on (e.g., insufficient fuel remaining, too far away). As a result, the handoff of the search occurs with an excessive gap.
    - ***Feedback about Collaborator Control Actions from Controlled Process***: *N/A*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment****:* The UAS on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
    - ***Construction***: Same as S-56.3.7. **In addition**:

[63] (B) Scenario: Team Lead has an inaccurate mental model of how long it takes the AuC to implement a command upon receiving it.

        - (Human-Machine) The control algorithms on the UAS are not compatible with how the TL specifies the <fix, search> task. (Same as S-47.1.7)
    - ***Initialization***: Same as S-56.3.7. **In addition**:
        - TL has a different belief about the current roles and responsibilities of the UAS. Feedback provided by the UAS team may be inadequate (e.g., missing, incorrect, delayed, ambiguous) for the TL to track which UAS was selected to hand over the search task to/from. As a result, the TL coordinates with or monitors the progress of the wrong UAS (i.e., not the UAS selected by the automation) and adjusts her/his actions regarding the search task incorrectly, in a way that leads to the gap.
    - ***Model Updates***: Elements of the UAS's process models are not adequately or consistently updated. ***Refinement***:
        - *Vertical Coordination (Control)*: TL tries to fine-tune the task handoff time to/from a UAS by adjusting the search task provided to the UAS team while the first controller <TL or UAS> is executing its part of the search. However, the automation does not assign a UAS to the slightly updated search task due to factors listed in S-43.7.3. This requires further action from the TL and the automation to resolve, which creates a delay and contributes to an excessive gap in the search.[64]
        - *Lateral Coordination (Communication)*:
            - Communications between the two controllers are not adequate to coordinate the handoff. (Same as S-47.1.7)
            - (Human-Machine) Lateral coordination between the TL and UAS is hindered by human-machine asymmetry in information semantics and timing. (Same as S-37.1.3)
        - *Lateral Coordination (Observation)*: (Same as S-48.2.7)
        - *Prediction*: (Human-Machine) Human and Machine experiences in training for this task handoff have evolved separately. The TL, which previously trained with humans, can anticipate when a human teammate is ready for a handoff based on implicit observations. Similarly, the machines may have been trained using simulation data, which does not precisely recreate human behavior. As a result, both controllers lack context or misinterpret certain cues that affect when and how to provide their respective commands in the handoff. (Similar to S-40.4.5)
    - ***Decision-Making***: Same as S-48.2.7
    - ***Capacity***: Same as S-40.4.5
- ***(Step 3: Collaborative Dynamic) Dynamic Membership***:
    - TL tasks the team based on an assumed set of UAS participants and an expectation that the automation will assign the task to a specific UAS well suited for the handoff. That UAS is then removed from the team, and the remaining participants

[64] (B) Scenario: The AuC is double checking the targets for accuracy, weapon type, etc. and takes too long before fixing on them, so AuC does not implement the command while targets are within range.

are unable to fulfill the tasking as intended by the TL. However, the TL does not adapt the tasking to overcome this change due to some of the factors listed above.
    - The UAS selected to hand off the task to/from the TL exits the team (e.g., retasked by Ground Station or a different team lead). This requires further action from the TL and automation to find a substitute, which contributes to an excessive gap in the task handoff.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity****: No new scenario conceived.* Changes in network topology contribute to the unsafe coordination factors listed above.

S-56.3.9 (Step 4: Other Factors): Members of the team hand off a search task in an unsafe way and create an excessive gap in the search due to factors beyond those explored in interactions between the controllers on the team. [TA]

- ***Inadequate Process Feedback****:* the first controller performing the search receives feedback that it temporarily misinterprets as no longer needing to search (e.g., it believes it has found the target of interest). It suspends its search, until it later changes its determination, and decides that more search is required. The time it takes to resume the search or coordinate the handoff with another UAS leads to an excessive execution gap.
- ***Unsafe Process Behavior****:* The target outmaneuvers the search in such a way that leads to the first controller performing the search to be unable to sustain it until the coordinated handoff time (e.g., forces high energy search maneuvers).

Abstraction 2a – Combinations of Control Actions Provided by the Team

Abstraction 2a – Type 1-2: Team provide / don't provide control actions

The following UCCAs explore how control actions provided or not provided by the team are unsafe in combination with one another. Causal scenarios consider how control actions and collaborative interactions internal to the team contribute to the UCCA. However, the number of possible internal control action combinations grows exponentially. In this case, the TL can individually task each of two UAS to fix and fire and task the team in general to search. So, there are $2^5 = 32$ possible combinations of control actions. By assuming the UAS are interchangeable, there are 20 unique combinations of internal control actions. Instead of considering every combination set of internal control actions, each UCCA is analyzed using the same *top-level scenarios* as listed in Abstract 2b – Type 1-2 UCCAs.

UCCA 1.1 (abstracted UCCA): The Team does not provide the fix, fire, nor search commands when there are mission tasks to execute [H3]

UCCA 1.1.1 (refined): TL, UAS1, and UAS2 do not provide fix, fire, and search commands

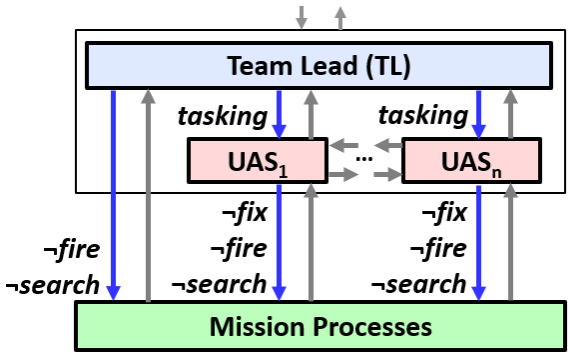


Figure A3-17. Control structure showing team not providing fix, fire, nor search

S-1.1.1 (Step 1: Top-Level Scenario 1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, the TL does not task anything when there are multiple tasks to be completed.

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL misinterprets direction from higher authorities that the UAS team should not be tasked to execute any mission tasks. Same as S-37.1.1. **In addition**:
    - Direction from higher authorities does not fit the description of the possible commands to task. For example, the TL is directed to only "prosecute" a target, which is ambiguous with regards to the fix, fire, search set of commands.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: (1) TL does not believe there are mission tasks to execute, (2) TL does not understand which mission tasks need to be executed, (3) TL does not believe there are teammates available to task, (4) TL believes that UAS(s) have already been tasked. Similar to S-37.1.1. **In addition**:
    - *Incorrectly Interpreted Feedback.* TL is overwhelmed by how many mission tasks and their completion status are presented to her/him, as well as how much information is provided regarding the UAS team. S/he is not able to make sense of it and does not provide the team any tasking.
    - *Feedback not available.* TL is so fixated on addressing one specific kind of task, such as defining a search area, that s/he loses sight of other tasks to perform, such as a coupled fix-fire task. This could occur if the feedback on the interface focuses only on one task at hand, and obfuscates others that need to be done.
    - *Misinterpreted Feedback:* The TL suspects there is a misconfiguration in the software version in one or more of the UAS (e.g., version number feedback, behavioral observations, …). As a result, the TL precautionarily chooses not to provide any taskings for lack of confidence they will be carried out as intended.
    - *Misinterpreted Feedback:* TL observes one or more UAS maneuvering in a way consistent with executing task(s), even though they are not. As a result, they do not provide necessary taskings.[65]

[65] Team Lead notices a set of UAV(s) begin following the target so he believes they are already tracking the target, so TL does not provide track target [or any other] command

- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the mission and the team capabilities, but still chooses not to task UAS(s). ***Refinement*** same as S-37.1.1 and S-43.7.5. **In addition**:
    - TL is too overwhelmed to adequately specify multiple mission tasks, allocate them to UAS, and ensure everyone is on the same page. As a result, the TL precautionarily chooses not to provide any taskings.[66]
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task UAS(s), but is unable to do so. Same as S-37.1.1. **In addition**:
    - TL tries to send multiple tasked commands at the same time. However, one of the commands is misspecified or exceeds the channel capacity. As a result, none of the commands are provided in the batch.
    - TL tries to send tasks one at a time (e.g., first tasks UAS1 to fix, next tasks the UAS team to search). However, every time a new tasking is provided, the system treats it as the most up-to-date set of taskings, and cancels/overrides previously issued ones (here, search cancels the fix).
    - One or more UAS have pre-programmed parameters (e.g., target <engage, search> location) that cannot be updated. Thus, the TL precautionarily chooses not to provide any taskings for lack of trust they will be carried out adequately.[67]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops***: *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*: *Not applicable. Top-level scenario focuses on control decisions from TL only.***
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.1. **In addition.**
    - TL does not provide multiple taskings at the same time out of concern that it would create a longer execution horizon that is more vulnerable to teammates dropping out of the team. For example, the TL tasked UAS1 and UAS2 to provide fix and fire tasks respectively, which will take some time to complete. The TL elects to not append a search task for the team that would take place after the fix-fire coupled task to avoid dealing with the UAS assigned to it no longer being available. However, UAS1 and UAS2 are unable to execute the fix-fire task, and instead of moving onto the search, remain untasked.
    - The overall set of mission tasks for the team to execute is highly dynamic. The set of mission tasks that need to be addressed change faster than the TL is able to specify and issue taskings to the UAS.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.1

S-1.1.2 (Step 1: Top-Level Scenario 2): TL tasks the UAS in a way that leads to unsafe collective control. Here, (1) the TL tasks multiple UAS to provide the same fix task (see S-37.1.2) or the same

[66] (B) Scenario: Team Lead confused the targets. He lost track of the correct target while trying to maneuver and tracked onto a different nearby target.
[67] (B) Scenario: The Team Lead implements the default values for identification.

fire task (see S-40.4.2), especially if there are multiple mission commands to perform. Also, (2) TL controller selection for each task is incorrect when considering all the controllers together.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.2. **In addition**:
  - Direction from higher authorities to task multiple UAS was intended to apply to search tasks only, and not fix and fire tasks. However, for various reasons (e.g., omission, miscommunication) TL interprets the direction as applying to all tasks.
  - Direction from higher authorities to task multiple UAS was intended to address multiple different targets, each with their own fix-fire tasks, but the TL interprets the direction as tasking multiple UAS per fix-fire tasks.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has one or more of the following inadequate process model variables: (1) UASi has not been tasked to <fix, fire> when it actually has, (2) multiple UAS will not cause hazardous effects if they <fix, fire> simultaneously, (3) the set of tasked UAS are capable of providing the set of tasked commands collaboratively, and (4) what the set of taskable controllers is at any given time. ***Refinement*** same as S-37.1.2. ***Refinement***:
  - *Incorrectly Interpreted Feedback.* TL simultaneously tasks UAS1 to provide a fix and the team to perform a search. UAS1 confirms it will execute a task, but does not specify which one. Given team observations (e.g., initial UAS1 maneuvering, lack of UAS2 announcing it has a task), the TL incorrectly assumes that UAS1 has taken on the search task. Therefore, s/he tasks UAS2 to provide the fix.
  - *Inadequate feedback:* Feedback to TL does not highlight gaps in the ability of two controllers to work together. For example, two UAS tasked to provide the coupled fix-fire commands are not able to do so together. This could be due to mutually imposed time or capability constraints.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-37.1.2. **In addition**:
  - Because TL is providing multiple taskings (fix, fire, and search), s/he tries to save time by sending out the same set of fix and search tasks to both UAS, with the intent to then change one of the UAS taskings from fix to fire. However, s/he gets distracted or disconnected from that UAS before the change can occur.
  - There are numerous tasks to provide to the UAS (e.g., multiple targets to engage). TL tasks UAS1 to fix and fire on Target 1. Then TL unintentionally tasks UAS2 to fix and fire on Target 1, thinking it was tasking it onto Target 2.[68]
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.2. **In addition**:
  - The tasking interface allows TL to specify tasks in formats that are incompatible with the UAS. For example, TL can task the team as a whole to provide to fix and fire, when TL should only specify UAS individually for those tasks.
  - Because the TL specifies a fix task to send out at the same time as a search task, the fix task gets sent out to all members of the team along with the search task. However, UAS team cannot allocate the fix to a UAS, so all UAS take it on.

[68] (B) Scenario: Team Lead provides the "fix on target" command for the right target, but there is a delay from an earlier fix on target command that the FMS acts on instead, so command is for a different target.

    - (Related to Dynamic Membership) As new mission tasks come and go, the TL's tasking system associates information with a mission process that has just disappeared to one that has just appeared. Thus, an outdated tasking provided by the TL for a mission process that is no longer relevant is reapplied to a new mission task, against the TL's intent.[69]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops***: *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*****:** *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.2. **In addition**:
    - TL tasks two UAS on the team to fix and search. UAS1 is assigned to fix and both UAS receive the search tasking, which is rated as higher-priority than the fix. UAS2 is assigned by the automation to search, but later drops out of the team (e.g., taken offline by Ground Station or attached to a different team). The autonomy reassigns UAS1 to the higher-priority search task, which now opens the fix task. Then UAS2 rejoins, and the TL proactively assigns it to provide the fix. The autonomy falsely assumes UAS2 will resume the search, and releases UAS1 back to its original task of providing a fix. Because two UAS are now assigned to fix, they both drop the task and no controller fixes nor searches.
    - The set of mission tasks is highly dynamic. By the time the TL tasks a UAS to fix or fire, that task is no longer relevant. However, the autonomy applies the previous tasking to the next target that appears, against TL intent (related to Unsafe Control Path). In parallel, TL tasks another UAS to fix or fire on the new target, which creates a conflict.[70]
- *Dynamic Connectivity:* Same as S-37.1.2.

S-1.1.3 (Step 1: Top-Level Scenario 3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, one UAS does not execute both a fix and fire task, two UAS both do not execute their respective fix and fire tasks in a coupled execution and/or no UAS performs the search task provided to the team. Emphasis is placed on instances when the UAS are tasked with both fix-fire and search tasks.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.3.
- ***(Step 2: Internal Control) Inadequate Process Model****:* See cognitive alignment
- *Inadequate Control Algorithm:* See cognitive alignment
- ***(Step 2: Internal Control) Unsafe Control Path****:* Same as S-37.1.3.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops****:* The way in which control loops are coupled for collaborative execution is shown in Figure A3-2 for the Fix-Fire tasks and in Figure A3-11 for the Search task. Same as S-37.1.3 and S-43.7.3.

---

[69] (B) Scenario: Team Lead is identifying old targets, so TL identifies the wrong targets.

[70] (B) Scenario: TL initially authorizes a fix on target command. There is a delay in another command from the TL to delete the command. AuC receives a fix on target command without authorization.

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another.
    - ***Construction***: The process models and/or control algorithms are not adequately or consistently built across the team to support collaborative control. ***Refinement***:
        - (Human-Machine) The control algorithms on the UAS are not compatible with how the TL specifies sets of multiple tasks. This prevents the controllers from accepting and prioritizing the execution of multiple tasks. For example, a TL tasks UAS to fix and fire respectively, and later task the team to perform a search. The UAS, unable to prioritize, drop the fix-fire tasks and execute the search. (Similar to S-37.1.3)
        - The control algorithms on the different UAS are not compatible with one another. As a result, the UAS lack the ability to coordinate in coupled fix-fire tasks or anticipate how an additional search task may take away some of the UAS resources. (Similar to S-37.1.3)
    - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
        - The way in which multiple tasks are specified is inconsistent relative to one another and prevents the UAS from coordinating on coupled task executions. For example, TL tasks UAS1 to fix, UAS2 to fire, and the UAS team to search. However, the way in which the fix task is specified (e.g., time window, priority) favors UAS1 performing it first, before considering the search. Conversely, fire task specification favors UAS2 performing it after the search is completed. As a result, no UAS perform any task. (Similar to S-37.1.3 & S-43.7.3)
        - The mission is highly dynamic and the set of mission tasks to execute changes constantly leading to the team having different beliefs regarding the overall joint process to control. This leads to teammates having different indices for available tasks (see Lateral Coordination by Communication).
        - UAS have different beliefs about how many and which UAS are participating on the team. This occurs due to dynamic membership, with UAS cycling on and offline, and dynamic connectivity that prevents timely distribution of this information. For example:
            - TL tasks UAS1 to fix and fire, and tasks the UAS team in general to search, expecting that UAS2 will be assigned to search. However, UAS1 is not aware of UAS2 and decides to take on the search task (which may have higher priority). As such, the fix and fire tasks are not executed.
        - UAS have different beliefs about the current roles and responsibilities of other controllers on the team. (Same as S-37.1.3 and S-43.7.3)
    - ***Model Updates***: Some elements of some of the process models are not adequately or consistently updated across the team. ***Refinement***:
        - *Vertical Coordination (Control)*: Same as S-37.1.3 and S-43.7.3. **In addition**:

- TL controls the coordination details between UAS1 (tasked to fix) and UAS2 (tasked to fire). However, s/he changes her/his mind and decides to provide the fire command instead of UAS2. This change of plan makes UAS2 drop its task and confuses UAS1 which does not know who to coordinate its fix task with.
- TL tries to optimize the planning horizon for two UAS tasked to jointly fix and fire followed by one UAS performing a search. S/he modifies the search task to start closer to the completion of the fix-fire coupled task. However, the change in parameters triggers the autonomy to replan and determine one of the UAS is now better suited to search first. All plans change, and the TL fights with the autonomy to resume the previous tasking.

▪ *Lateral Coordination (Communication)*: Same as S-37.1.3 and S-43.7.3. **In addition**:
- Controllers share a different set of indices for available tasks. UAS1 announces to the team that it is taking on task "x", which means different tasks to different controllers. Another controller announces a conflict and both controllers drop their tasks.
- UAS1 and UAS2 are both tasked to fix and fire respectively. However, UAS2 is also assigned to perform a search afterward. UAS2 shares coordination information about the wrong task (i.e., the search) with UAS1 and they are unable to come to consensus on how to execute the coupled task. Coordination delays also make UAS2 miss its search window.
- The information controllers use to plan and coordinate task executions is inconsistent. For example, a team of two UAS is assigned to fix and fire respectively, and also generally to search. They each compare slightly outdated state estimates shared by their partner to their own current states, which results in inconsistent plans. For instance, UAS1 (tasked to fix) believes it must perform the search after the fix-fire task. But UAS2 believes it must perform the search first, and then execute the fix-fire task. As a result, no tasks are executed.

▪ *Lateral Coordination (Observation)*: Similar to S-37.1.3, but more holistically: The controller tasked to fix (e.g., UAS1) observes the controller tasked for the coupled fire command (e.g., TL/UAS2) maneuvering in a way that is more consistent with executing the search. As a result, UAS1 incorrectly believes <TL, UAS2> will not fire, and it drops its plan to fix. Thus, <TL, UAS2> also drops its plan to fire.

▪ *Prediction*: Same as S-37.1.3 and S-43.7.3

○ ***Decision-Making***: Same as S-37.1.3 and S-43.7.3. **In addition**:

▪ (Similar to lateral coordination). Perturbations in inputs to the distributed decision-making process leads to the two UAS tasked to fix and fire

respectively and also to search as a team, to not reach consensus on the order of the tasks (i.e., search vs fix-fire first). Thus, no task is performed.

- The time it takes to coordinate two controllers in a fix-fire task or to select a UAS to search is too slow compared to the dynamics of the mission, with mission tasks coming and going. The TL proactively triggers a replan to prevent the planning process from stalling or planning on outdated information. However, that restarts the process without getting closer to controllers executing their tasks.
- The decision-making process does not have a mechanism to ensure convergence toward a feasible solution. It does not track previous bad decisions in planning to ensure they are not repeated.
    - For example, the control algorithm overemphasizes task prioritization and does not consider lower priority tasks that are feasible when higher-priority tasks are not. For instance, TL tasks two UAS to provide coupled fix-fire commands with high priority, and a search with low priority. The UAS are unable to reach consensus on a solution to execute the coupled fix-fire task, but their control algorithms fixate on trying to solve that problem rather than execute the search task which can be performed.
    - As another example, UAS1 is tasked to fix-fire by the TL, and the team is also tasked to search, which is assigned to UAS2. UAS1 determines it is not capable of performing the fix-fire tasks, so it drops the fix task and picks up the search as it is better positioned to execute it. This opens up the fix task, which is either retasked by the TL back to UAS1 or automatically re-picked up by UAS1, which is assigned to it. UAS1 now drops the search, UAS2 picks it up again, the cycle repeats itself, and nothing gets done.

- ***Capacity*:** Same as S-37.1.3. **In addition**:
    - The number of tasks and controllers that must be considered in a coordinated task allocation solution exceeds the computational limits of the controllers. The controllers are unable to form timely consensus due to the size of the state space that must be considered.

- ***(Step 3: Collaborative Dynamic) Dynamic Membership*:** Same as S-37.1.3. **In addition**:
    - TL tasks UAS1 to fix and fire, and the UAS team to search. UAS2 is therefore assigned the search. However, UAS2 has a problem that leads it to repeatedly change control modes and its participation status on the team. Every time UAS2 drops out, the algorithm replans and pulls UAS1 off the fix-fire task and assigns it to the search task. When UAS2 rejoins, the process reverses. As a result, there is "churn" and no UAS actually performs any task. (Similar to S-43.7.3)
    - The overall set of mission tasks for the team to execute is highly dynamic. The requirements to engage and search for targets change too rapidly for the assigned controller–control action pairing to execute anything.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity*:** Same as S-37.1.3 and S-43.7.3.

S-1.1.4 (Step 1: Top-Level Scenario 4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. *In the context of this UCCA, this high-level scenario is not applicable as no controller performs any task.*

S-1.1.5 (Step 1: Top-Level Scenario 5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, (1) TL does fire nor task a UAS to fire when a UAS provides a fix as tasked, or (2) TL does not search, nor task the UAS to search when there are untasked controllers. Emphasis is placed on instances when multiple tasks must be accomplished.

- ***(Step 2: Internal Control) Unsafe Control Input:*** *No scenarios conceived.*
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: TL incorrectly believes (1) s/he will perform certain tasks, (2) the UAS will perform certain tasks, (3) certain tasks do not need to be performed.
    - *Incorrectly Interpreted Feedback.* TL intends to perform the search her/himself and therefore does not task it out. But s/he misinterprets feedback of the process regarding how much time and fuel it will require for her to do it, and decides not to do it, but also does not retask it.
    - *Incorrect feedback.* TL previously tasked the UAS team to <fire, search>, but the tasking never reached the UAS (see Unsafe Control Path). However, the interface incorrectly displays that the command was sent, and thus, the TL believes the UAS are tasked and there is no need to do the task her/himself.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-37.1.1, S-40.4.5, S-43.7.5. **In addition**:
    - TL believes that different types of tasks may interfere with each other and decides to only execute one at a time. For example, TL intends to fire and UAS1 is tasked to fix, and there is a search task to perform. TL chooses not to task out the search to the UAS team, which UAS2 could carry out, in order to not interfere with the fix-fire coupled task. Similarly, TL may choose not to provide the search command when UAS are tasked to fix-fire. In either case, if the fix-fire command is not executed due to various reasons, no tasks are executed.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-1.1.1. **In addition**:
    - TL intends to provide the <fire, search> commands, but her/his <weapon system, search sensor system> equipment malfunctions and s/he is unable to do so.[71]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-40.4.5.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** Same as S-40.4.5 and S-43.7.5.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-40.4.5. **In addition**:
    - TL intends to perform the <fire, search> task, but s/he is too busy reassigning other tasks because of UAS dynamically joining and leaving the team or mission tasks coming and going.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-40.4.5 & S-43.7.5.

[71] (B) Scenario: MFC does not open the weapons bay, there is an electrical inference that causes the bay to unlock, or a missile is loose in the bay and pushed through the bay doors.

S-1.1.9 (Step 4: Other Factors): None of the controllers provide the fix, fire, or search commands due to factors beyond those explored in interactions between the controllers on the team.

- ***Unsafe Process Behavior:*** Same as S-37.1.9 and S-43.7.9. **In addition**:
  - The process is inundated with mission tasks to overwhelm the TL-UAS system. A malicious denial-of-service attack injects false tasks into the process.[72]

UCCA 10.5 (abstracted UCCA): The Team provides search but not the fix and fire commands when engaging a known target is higher-priority than searching and assets to engage are available. [H3]

UCCA 10.5.1 (refined): TL searches, and TL, UAS1, and UAS2 do not provide fix /fire commands

UCCA 10.5.2 (refined): UAS1 searches, and TL, UAS1, and UAS2 do not provide fix/fire commands

Other lower-priority UCCAs listed in Table A2-2 lead to the same. UCCAs 10.5.4-10.5.6 enumerate multiple controllers searching. UCCA 10.5.3 lists multiple UAS performing a fix, which is treated as not providing a valid fix due to interference, but is not relevant in this context.

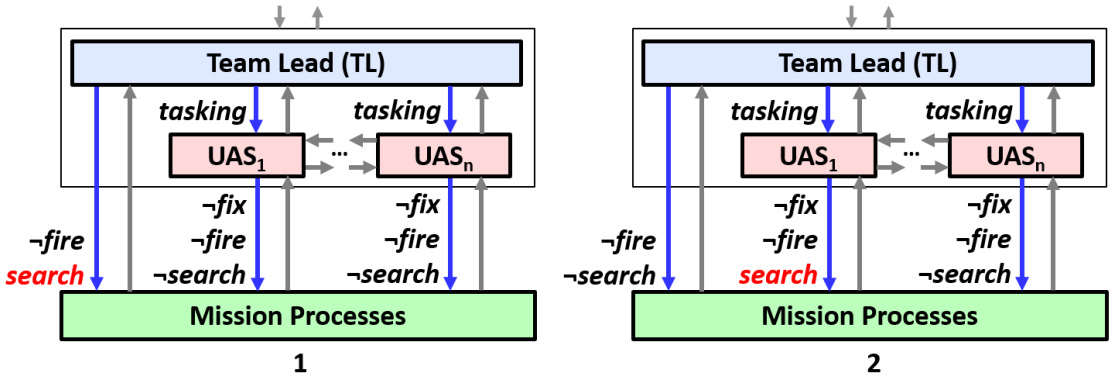


Figure A3-18. Control structure of team providing a search instead of the fix-fire commands

S-10.5.1 (Step 1: Top-Level Scenario 1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, TL does not task a UAS to fix and/or to fire (when the TL does not intend to fire), especially when TL tasks the UAS to search instead.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.1. **In addition**:
  - TL misinterprets direction from higher authorities that the search task is higher priority than engaging a target. (or vice versa)[73]
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has one or more of the following inadequate process model variables: TL believes (1) executing the search is higher-priority

[72] (B) Scenario: An attacker provides a false fix on target command to the FMS. An attacker provides false IFF responses. An attacker is spoofing pretend objects onto the FMS. An attacker places fake objects within the region.

[73] (B) Scenario: [Higher-authority] dis-authorization comes across as an authorization due to inaccuracies in the radio

than executing the fix-fire commands, (2) teammates are unavailable to fix or fire, (3) the fix- fire commands will be provided by the team. ***Refinement*** similar to S-37.1.1
    - *Misleading Feedback.* The search task as represented on the TL display is more visually prominent than the fix-fire tasks (e.g., it is larger in area) and as a result, draws more attention.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about mission priorities and the team capabilities, but still chooses not to task UAS(s) to support the fix-fire task. ***Refinement*** same as S-37.1.1 and S-40.4.5. **In addition**:
    - TL believes that part of the search can be performed, either by her/himself or by a UAS, enroute to performing the fix-fire task. Therefore, s/he tasks the team to search and intends to modify the tasking to the fix-fire task at the appropriate time. However, for a variety of reasons, this does not occur and the team only executes the search.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.1. **In addition**:
    - TL intends to task a UAS to fix to support the TL's fire task. However, the TL interface is inadequate, and instead, s/he unintentionally provides the wrong task (e.g., search task) to the team.[74]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops***: *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment***: *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - TL intends to task a UAS, however, the set of UAS participating fluctuates too much for the TL to have confidence that two UAS assigned to fix-fire will be still available to carry out the task. Instead, TL tasks the team to search to let the automation manages the changes in assignment due to dynamic availability. (Similar to S-37.1.1)
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.1

S-10.5.2 (Step 1: Top-Level Scenario 2): TL directs the UAS in a way that leads to unsafe collective control. The UAS recognize these conflicts and none of them execute their assigned mission tasks. Here, TL tasks multiple UAS to fix or to fire, especially when the TL also tasks the UAS to search. Same as S-1.1.2.

- **All factors**, same as S-1.1.2.
- ***(Step 2: Internal Control) Unsafe Control Input:*** In addition:
    - The TL misinterprets direction from higher authorities as a command to task all UAS on all mission tasks. This could occur if the intent is for the higher authorities to help the TL by directing Ground Stations to modify the existing taskings and fine-tune which UAS does what. Similar to S-1.1.2

[74] (B) Scenario: AuC interprets the "fix on target" command as a search for target command.

S-10.5.3 (Step 1: Top-Level Scenario 3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, UAS do not execute fix and/or fire tasks, especially when one of them executes a search task instead.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-10.5.1.
- ***(Step 2: Internal Control) Inadequate Process Model:*** Similar to S-1.1.3. **In addition**: TL incorrectly believes the UAS will prioritize certain tasks over others. ***Refinement***:
    - Missing Feedback: The UAS do not provide feedback on how tasks are prioritized.
    - Incorrect Feedback: The prioritization values displayed by the UAS are the ones that were provided by the TL as part of the tasking, but are not the values the UAS control algorithms actually use.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information on how UAS prioritize taskings and mission priorities, and still tasks the UAS to fix-fire and to search. Same as S-10.5.1. **In addition**:
    - TL tasks the UAS with default task priority values to get the UAS planning, but intends to modify those priority values prior to execution. Given the default priorities, the algorithm elects to do the search task first. The TL becomes busy and forgets to update the prioritization.[75]
    - TL unsuccessfully tries to influence the sequence of task execution using other operational "tricks". For example, the TL tasks the UAS to perform the fix/fire tasks first, before tasking the search. S/he incorrectly believes that by coordinating a solution on when to execute the coupled task will lock the UAS into this execution. However, as soon as the search task is provided, its specified priority parameters bring it to the top of the list, and one of the UAS drops its previous plans to execute the search. (or vice versa: switch search and fix-fire)[76]
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intends to task the UAS to prioritize the fix/fire commands, but is unable to do so.
    - It is faster and more "user-friendly" for the TL to simply copy over existing task templates, and modify some of the parameters as needed, rather than create a new tasking from scratch. S/he uses the same template for the fix-fire and the search tasks but forgets to update the priority parameters. The tasks are provided with equal priority, and the <search, fix-fire> task(s) are selected by the automation.[77]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** The way in which controllers mutually close control loops in task execution leads the team to execute the search and not the fix-fire coupled task.
    - Because of reasons explored in S-37.1.3 and S-40.4.5, the two controllers executing the fix-fire coupled tasks are unable to do so. There is logic for the automation to eventually stop trying and drop the task. This leads to one of the UAS becoming available to take on the search task and unavailable for the fix-fire tasks.

---

[75] (B) Scenario: Team Lead is pre-occupied with another task, so provides an identification command late.
[76] (B) Scenario: Team Lead is waiting for the UAV(s) to enter a specific position before applying the command so that certain UAV(s) will track the target.
[77] (B) Scenario: The Team Lead implements the default values for identification.

- Using the mutually closing search control loops concept described in S-43.7.3 - *Feedback about Controlled Process from Collaborators*: UAS1 and UAS2 are tasked to prioritize fix and fire tasks respectively, but the UAS team is also tasked to search with lower priority. UAS2 is in the vicinity of the search area and points its search sensor to gather feedback of opportunity. UAS1 incorrectly uses that feedback to update its belief that the search task is now higher in priority to execute first and changes its execution to focus on that task.

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another.
  - ***Construction***: Same as S-1.1.3 and S-44.8.3.
  - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
    - UAS receive different versions of the task specification by the TL, which causes them to evaluate the task prioritization differently. (Similar as S-37.1.3 and S-43.7.3)
    - The mission is dynamic and priorities change midway through the process of tasking UAS. For example, TL tasks the UAS team to search with medium priority and UAS1 to fix with low priority. Then the fix-fire task increases in priority so TL tasks UAS2 to fire with high priority but does not update UAS1. UAS1 chooses to search, and is not available to support UAS2 (or the TL) in their fire command.[78]
    - UAS have different beliefs about how many and which UAS are participating. This occurs due to dynamic membership, with UAS cycling on and offline, and dynamic connectivity that prevents timely distribution of information. As an example, TL tasks UAS1 to fix and UAS2 to fire. However, UAS1 is not aware UAS2 is part of the team, and decides the fix tasking is incomplete, and therefore moves on to the search task.
  - ***Model Updates***: Same as S-1.1.3.
  - ***Decision-Making***: Same as S-37.1.3 and S-43.7.3. **In addition**:
    - The distributed decision-making process is inherently more stable for certain types of tasks than others. Here, the search command only requires any one UAS to be available to execute it. Conversely, the fix-fire command may call for two specific UAS to work together and simultaneously, which is more difficult to satisfy. This can bias the output toward conducting searches. For example, UAS1 and UAS2 are respectively tasked to fix and fire, but the team is also tasked to search. UAS1 proposes a time for the coupled fix-fire task, which is rejected by UAS2. Further negotiating iterations may occur, but ultimately, UAS1 drops its fix task and takes on the search. Similarly, UAS2 drops its fire task.

[78] (B) Scenario: The algorithm determines that the task is not a priority, so Autonomous Controller does not implement a track target command.

        - Automated controllers employ a non-deterministic decision-making algorithm (e.g., Machine Learning based), which leads to unpredictable or inconsistent control and information outputs. Small variations in common information prevent the team from reaching consensus on the task prioritization and lead to a UAS deciding to search first.[79]
    - ***Capacity***: Same as S-37.1.3.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.3. **In addition**:
    - The behavior is different for different types of tasks if a UAS is assigned a task and momentarily drops out of the team. For example, the UAS is programmed to drop tasks it was specifically assigned to (e.g., fix or fire), but can still resume tasks generally assigned to the team (e.g., search). As a result, a UAS assigned to fix and/or fire with high priority may temporarily drop from the team, and rejoin by taking on the search command.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.3 and S-43.7.3.

S-10.5.4 (Step 1: Top-Level Scenario 4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, the UAS team is not tasked to search, but a UAS performs a search anyways instead of contributing to a fix-fire coupled task.

- ***(Step 2: Internal Control) Unsafe Control Input:***
    - The UAS team is tasked to search by another controller (e.g., the Ground Station). In this context, another controller may also increase the prioritization of a search task that was defined as low priority by the TL.[80]
    - A malicious actor floods the tasking system with invalid search tasks (e.g., cyber-attack). This obfuscates the real need to provide fix-fire commands.[81]
- ***(Step 2: Internal Control) Inadequate Process Model:*** see Cognitive Alignment. This high-level scenario is more focused on team cognition than that of any one controller.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:***
    - One of the UAS is in the process of executing a previously directed search task. TL now tries to task this UAS to execute a different command such as <fix, fire>. However, the UAS control algorithm does not allow a retask unless a specific direction is provided to abandon the current task. The TL does not (know to) direct the cancellation, and the UAS continues to search.
- ***(Step 2: Internal Control) Unsafe Control Path:***
    - TL intends to task the UAS to fix and/or fire, but unintentionally tasks the team to search instead. For example:[82]
        - TL has several tasks specified in advance of providing them to the UAS. TL then unintentionally selects the search task instead of the intended fix/fire task and hits send.

---

[79] (B) Scenario: AuC miscalculates the target's priority, so it does not implement task until not needed.
[80] (B) Scenario: An outside source provides a command with proper authorization, so implements a task.
[81] (B) Scenario: An attacker is spoofing pretend objects onto FMS, or places fake objects within the region.
[82] (B) Scenario: AuC interprets the "fix on target" command as a search for target command.

        - TL has several tasks specified in advance of providing them to the UAS. TL sends the first command in the list to the UAS (intended to be a fix-fire command). However, a different command on the list gets sent out. This could occur if (1) the intended command is mis-specified and is automatically skipped to the next one, or (2) the list of specified commands is displayed in a different order than that processed by the automation.
    - TL unintentionally tasks the search in addition to the intended fix-fire commands for similar reasons to those above. If the search is specified with higher priority, it could unintentionally lead UAS to execute the search instead of fix-fire tasks.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** The way in which controllers mutually close control loops in task execution leads the team to execute the search and not the fix-fire coupled task.
    - Using the mutually closing search control loops concept described in S-43.7.3 - *Feedback about Controlled Process from Collaborators*:
        - As UAS navigate untasked, they point their search sensors to collect information of opportunity, share the search feedback, and update the team's world view on the location of targets. One UAS senses information that it interprets as being such high-priority that it creates a search task, or updates search tasks preprogrammed in the UAS. This effectively creates a search task unintended by the TL, which one of the UAS takes on instead of carrying out the fix-fire coupled task as intended by the TL.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another.
    - ***Construction***: The process models and/or control algorithms are not adequately or consistently built across the team to support collaborative control. ***Refinement:***
        - The control algorithm on one or more UAS is not consistent with how the TL expects to execute the mission. For example, one of the UAS is configured to conduct a search as its default behavior if left untasked. This is a behavior that was unexpected to the TL and leads her/him to lose confidence in the ability of the UAS to contribute to fix-fire tasks. As a result, TL does not even task the UAS on the high-priority fix-fire task.[83]
    - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
        - One or more of the UAS has a preprogrammed search task that was created to relieve the burden of excessive task specification for the TL. However, the TL is not aware of this pre-initialized task. The TL specifies one new set of fix-fire tasks and provides "all active taskings" to the UAS team. The UAS(s) pre-initialized with a search task consider the search as

[83] (B) Scenario: The algorithm setup includes a default UAV option without the TL/GS knowledge

part of its planning, and in some cases, may assess that the pre-programmed search out-prioritizes the TL's only intended fix-fire tasks.[84]

- ***Model Updates***: Elements of the team's process models are not adequately or consistently updated. ***Refinement***:
  - *Vertical Coordination (Control)*: *No scenarios conceived*
  - *Lateral Coordination (Communication)*: A UAS is assigned to search but becomes disconnected from the team. In the meantime, TL provides new tasks, including a high-priority fix-fire coupled task. Now, UAS reconnects and shares the outdated search task with the team, which is interpreted as a valid task. As a result, one of the UAS takes on the search task instead of the intended fix-fire task.
  - *Lateral Coordination (Observation)*: See Prediction.
  - *Prediction*: (Human-Machine) The UAS have a model to anticipate the TL's next taskings so that the system can be better positioned to execute once the command is given. Given the observed behavior of the TL (e.g., maneuvering, information queries, …), the automation predicts that the next task to be issued will be a search. It queues this option to the TL for approval. The TL incorrectly approves this option without much thought, as s/he predicted the automation would recommend the fix-fire task. As a result, the UAS execute the search instead of the fix-fire task.[85]
- ***Decision-Making***: The process controllers use to decide what actions they provide is inadequate or inconsistent across the team.
  - The automation attempts to correct discrepancies in how the TL specifies the tasks, but that results in a solution that does not follow the TL intent. For example, the TL unintentionally tasks multiple (or all) UAS to provide the fix-fire commands. The automation recognizes that the only type of command that can be tasked to multiple UAS is the search. As a result, the UAS team reaches an incorrect consensus that the TL has tasked a search and carries it out instead of the intended fix-fire task.[86]
  - One UAS has a stale model of the open tasks and believes that the UAS team has been tasked to search. Its process to decide which UAS takes on the stale task conflicts with that of other UAS that correctly believe the task is no longer valid. The team correctly does not reach consensus on a task allocation for the search. However, the singled-out UAS then defaults to executing the search task itself, instead of dropping it.[87]
- ***Capacity***: The capacity of one of the controllers is inadequate to enable effective alignment of team cognition.

---

[84] (B) Scenario: AuC defaults to another object that looks similar, so it moves sensors to identify the wrong object [or perform the wrong task].
[85] (B) Scenario: Team Lead provides the fire command for the wrong target accidentally. TL was trying to lock on a target and instead authorized the UAV to fire at the target that it was tracking.
[86] (B) Scenario: AuC interprets the "fix on target" command as a search for target command.
[87] (B) Scenario: The algorithm stopped running and auto-selected the default UAV.

        - One UAS has a runaway internal process that ties up processing resources. It cannot update its model of tasks provided by the TL, and still believes a now stale search task is open. It continues to share this inadequate process model with the rest of the team. The task is eventually accepted by the team as real and is executed instead of the fix-fire task as intended.
- ***Dynamic Membership***: *No new scenarios conceived.*
- ***Dynamic Connectivity***: *Related to lateral coordination.*

S-10.5.5 (Step 1: Top-Level Scenario 5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL does not fire and a UAS provides a fix as tasked while no UAS is tasked to fire. Emphasis is placed on such instances when the TL provides the search command instead.

- ***(Step 2: Internal Control) Unsafe Control Input:*** *No scenarios conceived.*
- ***(Step 2: Internal Control) Inadequate Process Model***: Similar to S-40.4.5 and S-1.1.5: TL has the following inadequate process model variables: TL incorrectly believes (1) s/he will perform the fire tasks, (2) the UAS will not perform the fix task, (3) a UAS will perform the fire task, (4) the search task is higher-priority, and (5) providing a fix without firing is not unsafe. ***Refinement*** same as S-1.1.5. **In addition**:
    - *Feedback missing.* TL decides to fire her/himself, but at lacks feedback about her/his lack of capability as it pertains to the coupled fix-fire task (e.g., weapon system cannot couple to type of fix, cannot meet fix-fire timeline).[88] [89]
    - *Incorrect Feedback.* The TL's interface displays navigation parameters for the wrong task and directs the TL to search task instead of to fire. By the time s/he realizes this, it is too late to revert back to the original intended fire task.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-40.4.5. **In addition**:
    - TL believes s/he can provide the fire command while executing the search. S/he starts the search but becomes consumed by it and is unable to fire in coordination with the UAS providing the fix.
- ***(Step 2: Internal Control) Unsafe Control Path****: No new scenarios conceived.*
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:***
    - Using the mutually closing search control loops concept described in S-43.7.3 - ***Feedback about Controlled Process from Collaborators*****:**
        - TL intends to fire, and tasks a UAS to fix. Untasked UAS point their search sensors to collect information of opportunity, share the search feedback, and update the team's world view on the location of targets. One UAS senses information that is interpreted as being such high priority it changes the TL's task prioritization model. Rather than take the time to specify a search task, s/he decides to take on the search task immediately and leaves the fire task unaddressed. (Similar to S-10.5.4)

---

[88] (B) Scenario: Team Lead believes the target is within range because the AuC shows the target location within range even though the Team Lead receives a notification that it is out of range.

[89] (B) Scenario: TL provides firing command for wrong UAV because he has an incorrect understanding of the weapons on a specific UAV. FMS shows an inaccurate number of remaining missiles for a UAV.

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another.
    - ***Construction***: The TL and UAS have different logic to assume assignment of authority. For example, the TL defines three tasks: fix, fire, and search for a team consisting of the TL and two UAS. The TL tasks the fix to UAS1 and then assigns her/himself to conduct the search. The TL assumes (based on past experience, training, …) that assigning 2 of 3 tasks to 2 of 3 available controllers will implicitly assign the 3rd task to the 3rd controller. However, the logic on the UAS requires active tasking. No UAS is tasked to fire, and the TL conducts the search.
    - ***Initialization***: (Human-Machine) The process to specify a certain type of task (here search) for the machines is more difficult for the human than the process to specify other types of tasks (here fire) (e.g., more parameters, more complexity, more potential for semantic misalignment). Thus, the TL prefers to perform the search tasks her/himself, and tends to task out fire tasks. However, TL is unable to adequately task out the fire task (see S-37.1.1), but has already started the search task and becomes too consumed by it to recognize the need to retask the fire command.
    - ***Model Updates***: Elements of the team's process models are not adequately or consistently updated. ***Refinement***:
        - *Vertical Coordination (Control)*: *No scenarios conceived*
        - *Lateral Coordination (Communication)*:
            - UAS share parameters about different targets, without specifying which target they are describing. (Same as S-37.1.3). A UAS shares its intent to fire, which the TL misinterprets as an intent to fire on the designated target. As a result, the TL does not take further action to address the fire command and searches instead.
        - *Lateral Coordination (Observation)*: Same as S-37.1.3.
        - *Prediction*: The UAS that the TL intends to task to fire is currently disconnected from the team. Based on lost-communication procedures, the TL predicts it will reconnect with the team in time to task it. As such, the TL decides to perform the search. However, the UAS does not reconnect as predicted and it becomes too late to retask the fire command.
    - ***Decision-Making***: *No new scenarios conceived*
    - ***Capacity***: Same as S-40.4.5.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-40.4.5.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Related to lateral coordination.

S-10.5.9 (Step 4: Other Factors): Controllers on the team provide the search command, but not the fix and fire commands due to factors beyond internal team control and collaborative control.

- ***Inadequate Process Feedback:*** the tasked UAS have inadequate sensor feedback of their tasked mission process to execute the task. (Same as S-1.1.1.6)
- ***Unsafe Control Path to Process:*** Same as S-37.1.5
- ***Unsafe Process Behavior:*** Same as S-37.1.5.

UCCA 3.3 (abstracted UCCA): The Team provides the fire command but not the fix command when the target fired-on must be fixed. [H3, H5]

UCCA 3.3.1 (refined): TL fires, and UAS1 and UAS2 do not provide the fix command

UCCA 3.3.2 (refined): UAS1 fires, and UAS1 and UAS2 do not provide the fix command

UCCA 3.3.4 (refined): TL fires, and UAS1 and UAS2 both provide interfering fixes

UCCA 3.3.5 (refined): UAS1 fires, and UAS1 and UAS2 both provide interfering fixes

Other lower-priority UCCAs in Table A2-2 lead to the same outcome. UCCAs 3.3.3, 3.3.6-3.3.10 enumerate versions of the UCCA above, but with multiple controllers providing the fire command. Their factors are linked to the factors listed below and to those listed in S-42.6.

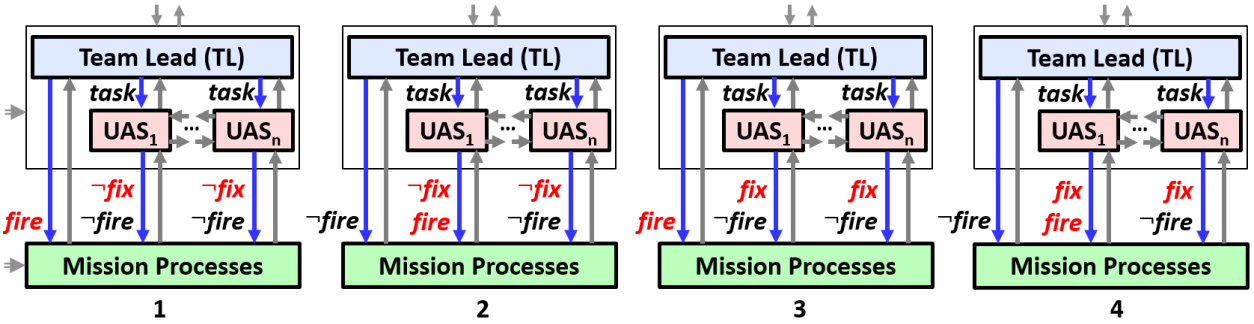


Figure A3-19. Control structures showing team providing fire but no fix commands

S-3.3.1 (Step 1: Top-Level Scenario 1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, (1) TL does not task a UAS to fix, but (2) TL fires or tasks a UAS to fire. Emphasis is placed on how (1) can happen without (2) and vice versa.

- ***(Step 2: Internal Control) Unsafe Control Input:***
  - TL misinterprets direction from higher authorities that the <fire, fix> command can be provided without a <fix, fire> command. (Similar to S-37.1.1)
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: (1) the <fire, fix> command can safely be provided without the coupled <fix, fire>, (2) no controller will provide the <fire, fix>, and (3) a controller will provide the <fix, fire>. ***Refinement*** similar to S-37.1.1, **in addition**:
  - *Delayed Feedback*. TL tasks a UAS to <fire, fix>, but the feedback is delayed or takes time to retrieve in the interface and still incorrectly shows the UAS as untasked. TL is unaware the UAS will actually perform the task, and thus, does not task/provide the <fix, fire> command to couple with it.
  - *Incorrectly Interpreted Feedback:* TL believes that her/his weapon is in an unarmed mode and cannot fire. Similarly, s/he believes the UAS tasked to fire is operating in a mode that will not accept the <fire, fix> tasking. As a result, s/he does not believe there is a need to task a UAS to provide the coupled <fix, fire> command.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about a controller intending to <fire, fix> and no controller tasked to provide the coupled

<fix, fire> command, but still chooses not to task the coupled command. ***Refinement*** same as S-37.1.1. **In addition**:

- TL misunderstands how the UAS operate. S/he incorrectly believes that if a controller is approved to <fire, fix>, the UAS automatically provides the coupled <fix, fire> command.[90]
- TL misunderstands how TL or UAS weapons systems operate. S/he incorrectly believes that if there is no fix, the weapon system will not fire. (Similarly, if no system announces intent to fire, the targeting system will not provide a fix)

- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.1. **In addition**:
  - TL specifies and sends both a fix and a fire tasking for the UAS. However, due to either a misspecification of one of the commands or due to inadequate communication channels, only one of the commands is provided.
  - TL does not intentionally task the fire command. S/he creates the task in the interface to be on standby for the right time. However, due to inadequate interaction with the interface, the task is sent to a UAS unintentionally.[91]
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops***: *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment***: *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
  - TL tasks a UAS that is then momentarily removed from the team. When the UAS rejoins, the TL incorrectly assumes it will resume its previous tasking. (Same as S-37.1.1)
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
  - TL tasks a UAS to <fix, fire, search>, but due to the dynamic topology, the tasking is not adequately routed to the intended UAS(s). (Same as S-37.1.1)

S-3.3.2 (Step 1: Top-Level Scenario 2): TL directs the UAS in a way that leads to unsafe collective control. Here, (1) TL tasks multiple UAS to provide the same fix task, and (2) s/he also fires or tasks a UAS to fire. Emphasis is placed on how (1) can occur (see S-37.1.2), and how (2) can occur with (1).

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL interprets a direction from higher authority as a request to fire even if multiple controllers provide a fix. (Similar to S-37.1.2)
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-37.1.2. ***Refinement***: TL incorrectly believes a controller cannot provide the <fix, fire> command even if tasked.

---

[90] (B) Scenario: TL does not provide fire [or fix] because he believes the UAV(s) can automatically fire upon identifying targets. TL assumes system works similar to other MUM-T systems where UAV(s) have automatic firing [or fix] capabilities.

[91] (B) Scenario: Team Lead provides the fire command for the wrong target accidentally. TL was trying to lock on a target and instead authorized the UAV to fire at the target that it was tracking.

- *Incorrect Feedback*: TL receives incorrect feedback from a UAS that its <targeting system, weapons system> equipment is inoperative.[92]

- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-37.1.2. **In addition**:
  - TL first tasks UAS1 to <fix, fire>. Then, s/he tasks UAS2 to provide the coupled <fire, fix> command and incorrectly believes that UAS2 does not have the capability to provide the same command as UAS1 (<fix, fire>). To save time, the TL simply appends the new <fire, fix> task, without deleting the UAS1 tasking, and sends both to UAS2.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.2. In addition:
  - TL tasks a UAS to provide the <fire, fix> command. Next, TL intends to task another UAS to provide the coupled <fix, fire> command. However, due to inadequacies in the interface, s/he unintentionally appends the 2nd task to the 1st task and submits both to the UAS.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops*:** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment*:** *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.2.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.2.

S-3.3.3 (Step 1: Top-Level Scenario 3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, (1) UAS do not execute the fix command as tasked and (2) the TL or a UAS fires. Emphasis is placed on how (1) can occur (See S-37.1.3), and how (2) can occur if (1) occurs.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.3. **In addition**:
  - TL interprets a direction from higher authority as authorization to provide the fire command, even if a fix is not possible.
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.3
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-37.1.3. **In addition**:
  - ***Feedback about Controlled Process from Collaborators***: The <TL, UAS> tasked to fire still receives feedback from the UAS tasked to fix, but which is not providing the fix, that the target needs to be fired on. ***Refinement***:
    - There is a misinterpretation of the meaning of the feedback. The UAS tasked to fix may be trying to say "target is not known to be destroyed" but that is interpreted as "fire on the target". (Human-Machine) This may be aggravated by human-machine asymmetry if the TL intends to fire.

[92] (B) Scenario: TL provides firing command for wrong UAV because he has an incorrect understanding of the weapons on a specific UAV. FMS shows an inaccurate number of remaining missiles for a UAV.

- ***Feedback about Collaborator Control Actions from Controlled Process***: The controller tasked to fire receives incorrect feedback from the target that leads it to believe the UAS is providing a fix. ***Refinement***:
  - A UAS provides a spurious fix (e.g., targeting system malfunction). The momentary feedback is misinterpreted as a sufficient fix to fire.[93]

  Note*: The UCCA previously analyzed in S-37.1.3 does not explicitly show the fire command. If the causal factors associated with this dynamic were not considered at that time, the analyst would consider them here.*

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
  - ***Construction***: Same as S-37.1.3. **In addition:**
    - The control algorithms executed by different controllers are not consistent with one another and lead to different determinations by the controllers on whether or not to execute a task. For example:
      - Even if controllers are configured similarly, the algorithm may apply a different decision-making process to execute the different types of tasks. For instance, a controller tasked to fix may determine that they are unable to execute the task based on inputs provided. However, the controller tasked to fire may determine differently that it is okay to continue.
      - Differences may also exist across how controllers are configured, leading to the same result. (Human-Machine) This can be exacerbated by asymmetry between the human TL and a machine UAS. (See S-37.1.3)
  - ***Initialization***: Same as S-37.1.3. **In addition:**
    - The process model of one of the controllers involved in the fix-fire coupled task is better initialized than the other. For example, the controller tasked to fire (which may include the TL) receives clear direction on how to execute their task. However, the UAS tasked to fix receives unclear information (e.g., due to inadequate communications, semantic mismatch). This contributes to only one controller performing the task.
  - ***Model Updates*****:** Same as S-37.1.3. **In addition**:
    - *Vertical Coordination (Control)*:
      - TL controls the coordination details between UAS1 (tasked to fix) and UAS2 (tasked to fire). However, s/he provides inconsistent information between the two and leads UAS1 to determine it cannot do its task, whereas UAS2 can (or vice versa).
    - *Lateral Coordination (Communication)*:
      - Coordination information sent by the UAS tasked to <fix, fire> to the controller tasked to the coupled <fire, fix> command is inadequate (e.g., incorrect, delayed, misinterpreted). It leads the

[93] (B) Scenario: AuC receives a target designation that it determines can lead to releasing a missile as long as parameters are satisfied, so AuC decides to release the weapon without a TL command

controller tasked to <fire, fix> to incorrectly believe the UAS can successfully provide the coupled task, and therefore the controller proceeds with the <fire, fix> command. (Human-Machine) This is exacerbated by human-machine asymmetry. (See S-37.1.3)

- Coordination information received by other controllers on the team lead the controller tasked to <fire, fix> to incorrectly believe the UAS can successfully provide the coupled <fix, fire>. For example, another UAS not involved in the fix-fire task communicates that a UAS is intending to <fix, fire>, but does not mention that it is engaging a different target.[94]

- *Lateral Coordination (Observation)*: The controller tasked to <fire, fix> observes the controller tasked to provide the coupled <fix, fire> command maneuver in a way consistent with that task and incorrectly believes it will execute it.
- *Prediction*: The controller tasked to <fire, fix> knows that a UAS has been tasked to provide the coupled <fix, fire> command, and as such predicts that the coupled task will be provided without checking for any feedback.
  - For example, TL tasks a UAS to fix and intends to fire her/himself. TL has confidence that the UAS will execute the fix as specified (e.g., based on past experience), and therefore does not bother to check if the fix feedback is adequate before firing.
  - As another example, the UAS tasked to fire becomes disconnected from the UAS tasked to fix. Given lost communications procedures and the execution context, the UAS tasked to fire predicts that the other UAS will provide a fix at a certain time, even though circumstances make that not possible. This contributes to the decision to fire even though the fix is not executed.

- ***Decision-Making***: Same as S-37.1.3. **In addition**:
  - Asynchronism in the distributed decision-making process leads some controllers to believe they have consensus when others do not. For example, the controller tasked to fix informs that its parameters are met to fix. After a delay, the controller tasked to fire determines its parameters are also met fires. However, by this time, parameters are no longer met for the other controller to provide the fix.
- ***Capacity***: Same as S-37.1.3. **In addition**:
  - One of the controllers has a runway internal process that makes it unable to coordinate with the team. Therefore, it does not process information from teammates that some tasks cannot be completed (e.g., <fix, fire>), and continues to press on with its own task.

- ***(Step 3: Collaborative Dynamic) Dynamic Membership***: Same as S-37.1.3. **In addition**:

[94] (B) Scenario: Team Lead provides the command early because he believes the right target is selected because AuC shows that it implemented the target designation command

    - The controller tasked to <fix, fire> is removed from the team. However, the controller tasked to provide the coupled <fire, fix> command is not aware of this loss. This contributes to proceeding with the coupled task anyway.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.3. **In addition**:
    - A 1st controller tasked to <fire, fix> is less connected to the team than a 2nd controller tasked to provide the coupled <fix, fire> command. The 2nd controller receives information faster and more adequately to inform it not to perform the coupled task than the 1st controller.

S-3.3.4 (Step 1: Top-Level Scenario 4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, (1) a UAS fires untasked, or when (2) a UAS provides a fix untasked and (3) another UAS provides a fix as tasked. Emphasis is placed on how (1) can occur (See S-41.5.3, S-41.5.4), and how (2) can occur (Similar to S-38.2.3) if (3) occurs.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-37.1.3
- ***(Step 2: Internal Control) Inadequate Process Model:*** see Cognitive Alignment. This high-level scenario is more focused on team cognition than that of any one controller.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** same as above.
- ***(Step 2: Internal Control) Unsafe Control Path:*** *No new scenarios conceived.*
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-38.2.3, but leads to an untasked UAS fix in addition to the tasked one.
    - ***Feedback about own control actions received from Collaborators***: An untasked controller <TL, UAS> receives feedback from a teammate that it misinterprets as a request to fire. (Similar to S-38.2.3)
    - ***Feedback about Collaborator control actions received from Controlled Process***: The controller tasked to fire receives incorrect feedback from the target that leads it to believe a UAS is providing a fix, and so it should provide the fire command, even if untasked. (Refined same as S-3.3.3)
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another.
    - ***Construction***:
        - The control algorithm on one of the UAS is programmed to allow it to provide a fix or a fire command, even if untasked, if certain parameters are met. This may be due to a software misconfiguration.[95]
    - ***Initialization***: Same as S-38.2.3.
    - ***Model Updates*:**
        - *Vertical Coordination (Control)*: *No scenarios conceived*
        - *Lateral Coordination (Communication)*: A UAS receives information from a teammate that it interprets as coordination associated with a <fire, fix>

[95] (B) Scenario: Auc algorithm is allowed to provide a root authorization if the original command does not involve firing a weapon or target designations, so AuC implements a task.

task. This leads the UAS to incorrectly believe it is tasked to provide that command. (Same as S-38.2.3)
        - *Lateral Coordination (Observation)*: Same as S-38.2.3
        - *Prediction*: A controller is temporarily disconnected from the team. However, a sequence of past communications and observations leads it to predict that if connected, it would be tasked to <fix, fire>. This influences it to take on the task, even though it is untasked.[96]
    - ***Decision-Making*:**
        - As part of distributed decision-making, one of the controllers introduces a new <fix, fire> task that is not part of the topic of coordination. This may occur due to a cyber-attack, or other refined reasons. Because the new task does not conflict with the topic of coordination, it is not rejected by the other controllers, and it becomes part of the team plan.[97]
    - ***Capacity***: No new scenarios conceived
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - A UAS was previously tasked to <fix, fire> by the TL, but was then temporarily removed from the team. As a result, the TL cancels the overall coupled fix-fire command. Then the UAS rejoins the team and continues its previous task even though this no longer follows the intent of the TL.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Related to Prediction.

S-3.3.5 (Step 1: Top-Level Scenario 5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL fires and does not task a UAS to fix. Factors associated with unsafe internal control are listed in S-3.3.1. Additional factors related to unsafe collaborative control are listed below.

> ***Note**: This UCCA may also consider when the TL fires and tasks another UAS to fire. By providing the fire command her/himself, the TL may cause UAS(s) that are properly tasked to both fix and fire to drop all their tasks together. Factors associated with this scenario are listed in S-42.6.5.*

- ***(Step 2: Internal Control) Unsafe Control Input, Inadequate Process Model, Unsafe Control Path:*** Same as S-3.3.1
- **(Step 2: Internal Control)** ***Inadequate Control Algorithm:***
    - TL chooses to fire or task a UAS to fire without a fix, under the incorrect anticipation that s/he will be able to task a UAS to fix in time before the weapon hits the target. (Reverse scenario also applies: TL tasks a UAS to fix without tasking one to fire under the incorrect anticipation that s/he will be able to task a UAS to fire in time before the fixed target compromises the mission)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-3.3.4
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement:***

---

[96] (B) Scenario: AuC does not receive a command from authorized provider, but it notices a UAV is following a target, so it decides to implement a search for target, which it was not provided.
[97] (B) Scenario: AuC determines an additional task is necessary to complete a command, and the TL provides the authorization without knowing why the task is necessary.

- ***Construction***: *No new scenarios conceived*
- ***Initialization***:
  - (Human-Machine) TL is unable to semantically specify the coupled fix-fire task to a level sufficient for her/him to have any confidence that a UAS will support her/him with the type of fix necessary. TL chooses to execute the fire task alone without tasking a UAS to fix.
- ***Model Updates***: *No new scenarios conceived (no tasking takes place)*
- ***Decision-Making***: *No new scenarios conceived (no tasking takes place)*
- ***Capacity***: TL workload is too heavy to specify a fix tasking for a UAS. S/he chooses to fire alone without of tasking a UAS to fix. (Similar to S-40.4.5)

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.1
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.1

S-3.3.9 (Step 4: Other Factors): Controllers on the team fire, but do not provide the fix due to factors beyond those explored in interactions between the controllers on the team.

- ***All factors:*** Same as S-37.1.9.

UCCA 2.2 (abstracted UCCA): The Team provides the fix command but not the fire command when a target may compromise the mission if fixed but not fired on. [H3]

UCCA 2.2.1 (refined): UAS1 fixes, and TL, UAS1, and UAS2 do not provide fire command

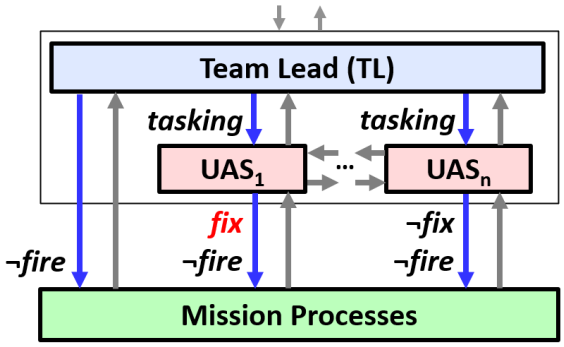


Figure A3-20. Control structure showing team providing fix but no fire command

***Note**: UCCA 2.2 involves many factors involved in UCCA 3.3 as both involve the team providing either the fix or fire command and not the other. As such, the following analysis largely points back to S-3.3.*

S-2.2.1 (Step 1: Top-Level Scenario 1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, (1) the TL does not task a UAS to fire (and does not intend to fire), but (2) the TL tasks a UAS to fix. Emphasis is placed on how (1) can occur if (2) occurs and vice versa.[98]

- ***All factors:*** Same as S-3.3.1.

[98] (B) UCA 15: Team Lead provides the track target command early and exposes the mission (H3)

S-2.2.2 (Step 1: Top-Level Scenario 2): TL directs the UAS in a way that leads to unsafe collective control. Here, TL (1) tasks multiple UAS to provide the same fire task and (2) also tasks a UAS to fix. Emphasis is placed on how (1) can occur (see S-40.4.2), and (2) can occur with (1).

- *All factors:* Same as S-3.3.2.

S-2.2.3 (Step 1: Top-Level Scenario 3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, (1) a UAS does not execute the fire command as tasked and (2) a UAS provides a fix. Emphasis is placed on how (1) can occur (See S-40.4.3), and (2) can occur if (1) occurs.

- ***(Step 2: Internal Control) Unsafe Control Input, Inadequate Process Model, Inadequate Control Algorithm, Unsafe Control Path:*** Same as S-3.3.3.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment, Dynamic Membership, Dynamic Connectivity:*** Same as S-3.3.3.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-40.4.3, but Machine-Machine interactions. **In addition**:
    - ***Feedback about own control actions received from Collaborators***: The UAS tasked to fix receives feedback from the UAS tasked to fire, but which will not fire, that it interprets as a request to fix. ***Refinement*** similar to S-3.3.3:
        - There is a misinterpretation of the meaning of the feedback. The UAS tasked to fire provides the feedback as factual information without implying an intent to fire or a need to fix. (Similar to S-3.3.3)
    - ***Feedback about Collaborator control actions received from Controlled Process***: *No scenarios conceived.*

  Note: *The UCCA previously analyzed in S-40.4.3 does not explicitly show the fix command. If the causal factors associated with this dynamic were considered at that time, the analyst would consider them here.*

S-2.2.4 (Step 1: Top-Level Scenario 4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, a UAS provides an untasked fix (Similar to S-38.2.3).

- ***(Step 2: Internal Control)*** *Unsafe Control Input, Inadequate Process Model, Inadequate Control Algorithm, Unsafe Control Path:* Same as S-3.3.4
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment, Dynamic Membership, Dynamic Connectivity:*** Same as S-3.3.4
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:***
    - ***Feedback about own control actions received from Collaborators***: An untasked UAS receives feedback from another controller that the fix is inadequate to fire. It misinterprets this as a request to provide a fix, even though the feedback may not be associated with any intent to fire. Similar to S-38.2.3
    - ***Feedback about Collaborator control actions received from Controlled Process***: *No scenarios conceived.*

S-2.2.5 (Step 1: Top-Level Scenario 5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, (1) the TL does not fire (and does not task a UAS

to fire), and (2) TL tasks a UAS to fix. Emphasis is placed on how (1) could occur if (2) occurs, and vice versa.

- ***(Step 2: Internal Control) Unsafe Control Input, Inadequate Process Model, Inadequate Control Algorithm, Unsafe Control Path:*** Same as S-3.3.1
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops, Cognitive Alignment, Dynamic Membership, Dynamic Connectivity:*** Same as S-40.4.5

S-2.2.9 (Step 4: Other Factors): Controllers on the team provide the fix, but do not fire due to factors beyond those explored in interactions between the controllers on the team.

- ***All factors:*** Same as S-3.3.9.

UCCA 11.6 (abstracted UCCA): The Team provides the fix and fire commands but not the search command when searching for another target has higher priority [H3]

UCCA 11.6.1 (refined): TL fires, UAS1 fixes, and TL, UAS1, UAS2 do not search

UCCA 11.6.3 (refined): UAS1 fixes, UAS2 fixes, and TL, UAS1, UAS2 do not search

UCCA 11.6.5 (refined): UAS1 fixes and fires, and TL, UAS1, UAS2 do not search

Other lower-priority UCCAs listed in Table A2-2 lead to the same outcome. UCCAs 11.6.2, 11.6.4, 11.6.6-11.6.8 enumerate multiple controllers performing the fix and or fire tasks.

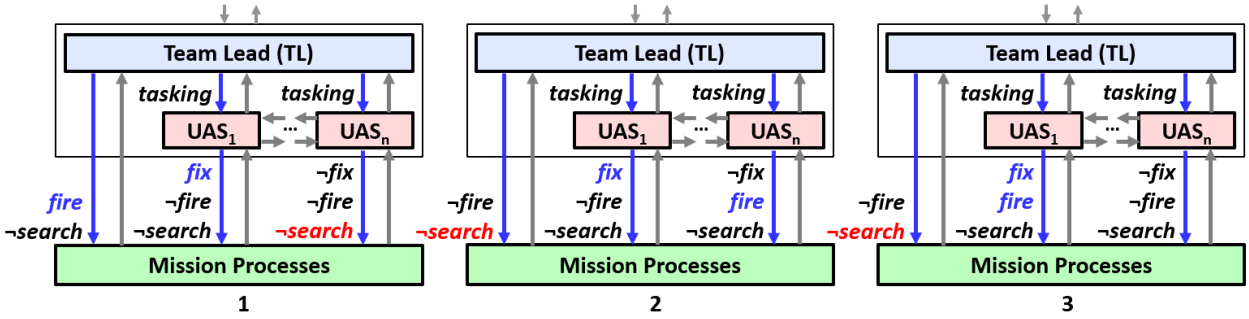


Figure A3-21. Control structure of team providing a search instead of the fix-fire commands

S-11.6.1 (Step 1: Top-Level Scenario 1): TL does not direct the UAS as necessary for the team to execute safe collective control of the search instead of the fix-fire commands. Here, the TL does not task a UAS to search (if the TL does not intend to search), and the TL tasks the UAS to fix-fire instead.[99]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-10.5.1
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-10.5.1, but search and fix-fire reversed.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-10.5.1, but search and fix-fire reversed.

[99] (B) UCA 7: TL provides "fix on target" command before AuC has an identification of the target (H3)

- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-10.5.1, but search and fix-fire reversed.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - TL intends to task the UAS to search, however, the set of participating UAS fluctuates too much for the TL to have confidence that the automated tasking algorithm will not churn. Instead, TL tasks UAS to fix and fire so that s/he can handpick controllers s/he is confident will not fall out. (Similar to S-10.5.1)
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.1

S-11.6.2 (Step 1: Top-Level Scenario 2): TL directs the UAS in a way that leads to unsafe collective control. The UAS recognize these conflicts and none of them execute their assigned mission tasks. Here, TL tasks multiple UAS to fix or to fire, especially when the TL does not task the UAS to search instead.

- ***All factors*** *covered by scenarios in* S-1.1.2, S-3.3.2, and S-11.6.1

S-11.6.3 (Step 1: Top-Level Scenario 3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, no UAS executes the search task, especially when one or more of them participate in a fix-fire task instead.

- ***(Step 2: Internal Control) Unsafe Control Input, Inadequate Process Model, Inadequate Control Algorithm, Unsafe Control Path:*** Same as S-10.5.3.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops****:*
    - ***Feedback about Controlled Process from Collaborators***: Same as S-10.5.3.
    - ***Feedback about Collaborator Control Actions from Controlled Process****: N/A.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment****:* The controllers on the team have inadequate cognitive alignment relative to one another.
    - ***Construction***: Same as S-1.1.3 and S-44.8.3. But in reverse order, leading the UAS to drop the higher-priority search task in favor of executing the lower fix-fire task.
    - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
        - UAS receive different versions of the task specification by the TL, which causes them to evaluate the task prioritization differently. (Similar to S-37.1.3 and S-43.7.3)
        - The mission is dynamic and priorities change midway through the process of tasking UAS. Same as S-10.5.3, but results in UAS dropping the search task in favor of fix-fire tasks.
        - UAS have different beliefs about how many and which UAS are participating. As such, TL tasks UAS team to search with high priority, and UAS1 to fix with low priority. UAS1 incorrectly believes another UAS is better positioned to execute the search task, and assumes it will be assigned to it, so UAS1 takes on the fix command instead. However, there is no other UAS, so the priority search is not executed. (Similar to S-10.5.3)
    - ***Model Updates*****:**
        - *Vertical Coordination (Control)*: Same as S-43.7.3

        - *Lateral Coordination (Communication)*: Same as S-43.7.3, **in addition:**
            - Teammates change and share inconsistent task execution parameters. For example, TL tasks UAS to search with medium priority, UAS1 to fix with low priority, and UAS2 to fire with low priority. The search gets assigned to UAS1. UAS2, which is now not responsible to search, updates its own task priority for fire to high. However, it then shares this updated information, which leads UAS1 to believe the coupled fix task should be high priority. Therefore, it drops the higher-priority search task.
        - *Lateral Coordination (Observation)*: Same as S-37.1.3
        - *Prediction*: No new scenario conceived.
    - ***Decision-Making*: Same as S-37.1.3 and S-43.7.3. In addition:**
        - The distributed decision-making process biases coupled tasks over individual tasks. For example, the UAS are tasked to search with medium priority and fix and fire each with low priority. However, the decision-making process aggregates the values of the fix and fire tasks and determines that together they are higher-priority than the search task.
    - ***Capacity*:**
        - Some tasks require higher capacity to process and are thus less likely to be executed by controllers that can only process simpler tasks. For example, the automation tasks a UAS to search. However, executing the search involves computing an optimal route subject to constraints, which takes longer to compute for a UAS than fix-fire tasks. As a result, the UAS can reach a feasible solution for the fix-fire tasks and may become committed to them before it knows whether or not it can take on the search. This biases the UAS to fix-fire over searching, even if the search is of higher priority.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-10.5.3.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S-11.6.4 (Step 1: Top-Level Scenario 4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, UAS(s) are not tasked to fix-fire but do so anyways instead of performing the tasked search.

- ***All factors:*** Same as S-10.5.4, but leads to fix-fire tasks instead of the search task.

S-11.6.5 (Step 1: Top-Level Scenario 5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL does not search while no UAS is tasked to search. Emphasis is placed on when TL fires and tasks a UAS to fix instead.

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL misinterprets direction from higher authorities that s/he should always prioritize fix-fire tasks over searches, or that s/he should always prioritize providing coupled tasks to support UAS.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: TL incorrectly believes (1) s/he will perform the search task, (2)

the UAS will not perform the fix task, (3) a UAS will perform the search task, (4) the fix-fire tasks are higher-priority. ***Refinement*** same as S-1.1.5.

- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-10.5.5, but results in the fire command and not search.
- ***(Step 2: Internal Control) Unsafe Control Path:*** TL intended to task a UAS to fire and perform the higher-priority search her/himself. However, due to factors listed in S-37.1.1, her/his command does not reach the UAS, and s/he is forced to fire instead of searching because a UAS is already tasked to fix.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Due to factors described in S-40.4.5, the UAS that was tasked to fire is unable to do so. As a result, the TL is forced to fire instead of searching because a UAS is already tasked to fix.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** Same as S-10.5.5, but leads to TL believing a UAS will execute a search. In addition:
    - ***Model Updates*:**
        - *Prediction*: TL tasked UAS1 to fix and UAS2 to fire, both with high priority. TL also tasked the UAS team to search with low priority and intends to search her/himself. UAS2 becomes disconnected from TL. TL predicts based on lost-communications procedure, that UAS2 will default to taking on the non-coupled search task and drop the fire task. As such, TL changes plans and decides to fire her/himself in collaboration with UAS1 providing the fix. However, UAS2 is still connected to UAS1 and still intends to provide the fire command. Now no controller provides the search command. Similar to S-10.5.5.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-40.4.5.
    - TL anticipates that the UAS tasked with providing the fire may exit the team (e.g., retasked by Ground Station). As such, TL holds off on searching so that s/he can be available to execute the fire if needed. Similar to S-40.4.5.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Related to Prediction.

S-11.6.9 (Step 4: Other Factors): Controllers provide the fix-fire commands, but not the search due to factors beyond those explored in interactions between the controllers on the team.

- ***All factors:*** Same as S-10.5.9

UCCA 4.4 (abstracted UCCA): The Team provides the fix and fire commands when the target fixed on is different from the target fired on [H3, H5]

UCCA 4.4.1 (refined): TL fires and UAS1 fixes

UCCA 4.4.5 (refined): UAS1 fixes and UAS2 fires

Other lower-priority UCCAs listed in Table A2-2 lead to the same outcome. UCCAs 4.4.4 and 4.4.7 involve multiple controllers performing the fix task. UCCAs 4.4.3, 4.4.6, 4.4.8-4.4.12 enumerate multiple controllers performing the fire task. UCCA 4.4.2 consists of one UAS fixing and firing on a different target, which is not a collaborative control problem.

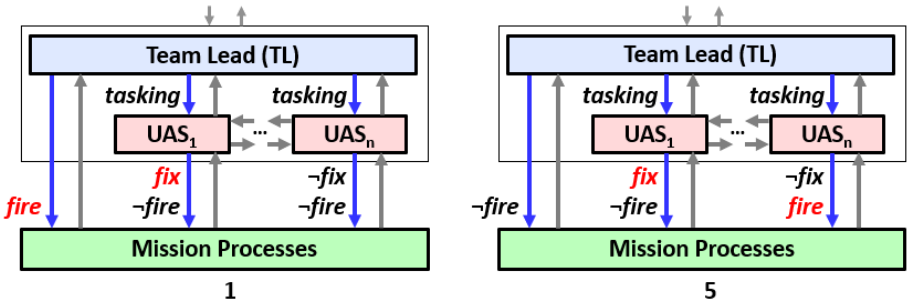


Figure A3-22. Control structure showing team the fix & fire commands on different targets

S-4.4.1 (Step 1: Top-Level Scenario 1): TL does not direct the UAS as necessary for the team to execute safe collective control. Here, (1) TL does not task a UAS to fix (or fire) and (2) TL does task the coupled fire (or fix) task (while not intending to fire her/himself). It also emphasizes how despite the missing taskings, the UAS provides the missing task, but not in a way that is aligned with the coupled task.

> ***Note**: this scenario is deprioritized because of the contrived combinations of the team interactions it necessitates (many different things need to go wrong together).*

S-4.4.2 (Step 1: Top-Level Scenario 2): TL directs the UAS in a way that leads to unsafe collective control. Here, (1) TL tasks a UAS to fix on a target and another to fire on a different target, and (2) the UAS carry out the unsafe tasking.[100]

- ***(Step 2: Internal Control) Unsafe Control Input:*** No new scenarios conceived
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: TL believes (1) the target for the UAS1 task is the same as for UAS2, (2) other tasks fix-fire tasks will be executed for each of the two different targets.
  - *Misinterpreted Feedback.* The interface shows that targets have been assigned tasks for the team to perform. But it does not show what type of tasks they are nor if the combined set assigned per target is adequate.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information that the fix and fire tasks are specified for different targets, but still chooses to issue them.
  - TL incorrectly believes that providing the tasks that way will lead to UAS1 and UAS2 fix-fire on both targets. S/he believes the tasking authorizes the UAS to do the type of task, authorizes the targets to be engaged, and that the UAS will coordinate how all that is accomplished.
  - TL incorrectly believes that each UAS will automatically provide both the fix and fire commands for each target. (Same as S-37.1.1)
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.1. In addition:

[100] (B) UCA 3: Team Lead provides "fix on target" command for the wrong target (but still an enemy target) (H5); UCA 18: Team Lead provides firing command for the wrong target (H5)

    - TL intends to task the UAS to fix and fire on the same target. However, the TL interface is inadequate and leads her/him to unintentionally specify different targets for the two tasks.[101]
        - This could be the result of the interface providing predictive target specification for the task (e.g., given the location of the UAS), which the TL approves without awareness that the target is incorrect.
    - TL does not intend to provide any tasks yet but specifies them ahead of time as placeholders. However, the interface treats the inputs as taskable, and provides them to the UAS.[102]
    - TL intends to task both UAS onto both targets. However, due to Unsafe Control Path factors in S-37.1.1, some of the fix and fire tasks are not provided.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** *Not applicable. Top-level scenario focuses on control decisions from TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Similar to S-37.1.1.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S-37.1.1.

S-4.4.3 (Step 1: Top-Level Scenario 3): TL directs the UAS adequately, but some of the UAS do not execute the directions properly, which leads to unsafe collective control. Here, (1) TL tasks the UAS to fix and fire on the same target, but the UAS provide those commands on different targets, or (2) TL tasks UAS to both commands (fix and fire) on two different targets, but the UAS only provide one of each on different targets.[103]

- ***(Step 2: Internal Control) Unsafe Control Input:*** The UAS assignments are overridden by other controllers. For example, the Ground Station changes the target specified for one of the two UAS. Or, the Ground Station deletes the additional fix and fire tasks that would have enabled both targets to be engaged. (Similar to S-37.1.3)
- ***(Step 2: Internal Control) Inadequate Process Model:*** See Cognitive Alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See Cognitive Alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:*** The UAS intend to fix-fire on the same target, but one of the UAS has a <targeting system, weapon system> malfunction, and the effects of the command are provided to the wrong target. (Same reason applies to the TL firing on the wrong target)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:***
    - ***Feedback about Controlled Process from Collaborators*:** The feedback either of the two UAS tasked to fix or fire gets from the other leads it to <fix, fire> on a different target.
        - UAS1 declares it is ready to start the <fix, fire> command on a target, but does not specify which target it is acting on to its collaborator (UAS2, TL),

[101] (B) Scenario: the command only contains an approximate location of the target, but there are several potential targets within range and the FMS selects wrong one, and command is fixed on a different target.
[102] (B) Scenario: FMS interprets an identification command as a fix on target command.
[103] (B) UCA 36: Autonomous Controller releases a missile for the wrong target (H5)

which assumes it is for the same target they intend to act on. Alternatively, UAS1 specifies a target that is different from the target it intends to act on.
        - UAS1, tasked to fix, provides feedback to UAS2, tasked to fire, regarding the status of multiple different targets, including Target 2 which is not yet destroyed. UAS2 misinterprets that feedback as a request to fire on Target 2, even though UAS1 ultimately settles on fixing Target 1 as tasked.
    - ***Feedback about Collaborator Control Actions from Controlled Process:***
        - Another UAS provides a fix for a different target (Target 2) as part of a separate tasking. However, that fix is interpreted by the controller tasked to fire as the fix it was seeking for Target 1.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
    - ***Construction***: No new scenarios conceived.
    - ***Initialization***: Some elements of the process models are not adequately or consistently initialized across the team. ***Refinement***:
        - UAS receive different versions of the task specification by the TL, which causes them to evaluate the designated target differently. For example, TL initially specifies the coupled task for Target1, but then replans and specifies it for Target2, but only a subset of the UAS receive the update. (Similar to S-37.1.3)
    - ***Model Updates*:**
        - *Vertical Coordination (Control)*:
            - TL specifies separately the target for each of the UAS. However, there is an unintentional mismatch between the two targets specified, due to various factors (e.g., inadequate interface, operator error, dynamic targets).
        - *Lateral Coordination (Communication)*:
            - Communications between the UAS assigned to fix and the controller assigned to fire are not adequate to coordinate on the target that is intended in the tasking. Bits are swapped in messages, leading one of the controllers to misinterpret the target.
            - A UAS is temporarily disconnected and its model of adynamic targets diverges from that of the team. It eventually associates one of the targets with the wrong identifier (e.g., swaps two targets that were maneuvering in proximity to each other). When the UAS reconnects, it contributes its divergent model to the team, which contributes to some of its teammates also swapping targets. This leads to UAS1 (tasked to fix) and UAS2 (tasked to fire) having a different model of the target they are jointly engaging.
        - *Lateral Coordination (Observation)*:
            - UAS1 (or TL) tasked to <fix, fire> observes UAS2 (or TL), which is tasked to the coupled <fire, fix> task, maneuvering in a way consistent with acting on a different target. This influences UAS1

(or TL) to change the target over which it acts, even though that is not what UAS2 (or TL) is intending to do.

- *Prediction*:
  - UAS1 is tasked to fix on Target1, and is aware that UAS2 is tasked to fire (although UAS2's target is not explicitly specified). UAS1 incorrectly predicts from its own tasking that UAS2 will fire on Target1 and proceeds with executing its task.

- ***Decision-Making*:** Same as S-37.1.3 and S-43.7.3. In addition:
  - The distributed decision-making process does not consider all the information shared in coordination. For example, UAS1 and UAS2 openly communicate that they are ready to execute their tasks on Targets 1 and 2 respectively. However, their algorithm does not actually consider the target specified by the other controller and instead plans for the target it assumes the other controller is controlling.
  - The distributed decision-making process correlates other parameters beyond the target index to reach consensus. For example, UAS1 and UAS2 have multiple fix-fire tasks to support each other. The decision-making process looks for the closest match between any two tasks to determine a coupling solution. Because of other factors (e.g., inconsistent model initialization and updates) UAS1 fixing Target1 is assessed as a better match with UAS2 firing on Target2 than with UAS2 firing on Target1.
  - There are too many tasks to fix and fire on defined for the UAS to determine the optimal task execution order. The controllers use approximations and heuristics (such as greedy task allocation) to determine their next course of action. While this type of decision-making is efficient, in some cases it leads to UAS coupling tasks that should not be coupled.[104]

- ***Capacity***: *No new scenario conceived.*

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-10.5.3, but leads to differences in which target to act on.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S-4.4.4 (Step 1: Top-Level Scenario 4): TL adequately does not direct the UAS to provide certain commands, but some of the UAS provide them anyways, which leads to unsafe collective control. Here, one UAS is tasked to fix (to support the TL that intends to fire), and another UAS that is not tasked to fire does fire on a different target. Emphasis here is placed on why that untasked UAS would fire on a different target.[105]

- Factors associated with an untasked UAS firing are explored in S-41.5.4 and S-3.3.4. *No additional scenarios conceived.*

---

[104] (B) UCA 32: Autonomous Controller implements a task before completing an algorithm to determine the optimal task completion

[105] (B) UCA 36: Autonomous Controller releases a missile for the wrong target (H5)

S-4.4.5 (Step 1: Top-Level Scenario 5): TL control actions to the controlled process are unsafe in combination with how it directs the UAS. Here, TL fires on a target that is different than the one s/he tasked a UAS to fix.[106]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-3.3.1.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: TL believes that the UAS will provide the fix command on the same target the TL intends to fire on. ***Refinement***:
    - *Incorrect Feedback:* TL tasks UAS1 to fix Target1, UAS1 provides incorrect feedback that it will fix Target2 (e.g., stale information from previous tasking) even though it plans to fix Target1. TL now changes plans and fires on Target2.[107]
    - *Missing Feedback:* TL tasks UAS1 to fix Target1, UAS1 misinterprets the command and fixes Target2, but only replies that it is providing the fix, and not which target it is fixing. TL now fires on Target1 without knowing UAS1 is fixing Target2.[108]
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-3.3.1.
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-4.4.3
- ***Mutually Closing Control Loops:***
    - ***Feedback about Controlled Process from Collaborators*:** Same as S-4.4.3.
    - ***Feedback about Collaborator Control Actions from Controlled Process:***
        - One of the UAS provides a spurious fix of another target due to a targeting system malfunction. The momentary feedback is misinterpreted by the controller tasked to fire (TL, UAS2) as the fix to fire on. Similar to S-3.3.3
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
    - ***Construction***:
        - The control algorithms executed by different controllers are not consistent with one another, and leads to different target determinations.
            - For example, the target selected may be specified as "the largest" target from a set. However, "largest" may be interpreted differently (e.g., largest in physical size, largest in signature, …).
    - ***Initialization***:
        - (Human-Machine) TL and the UAS have a different initial representation of the target due to semantic mismatches between how the TL describes it in the tasking, and how the UAS interprets it (related to construction).
    - ***Model Updates*:** Same as S-37.1.3. In addition:
        - *Vertical Coordination (Control)*: TL provides coordination specifics for how the UAS should provide the fix that is more aligned with the UAS's

[106] (B) UCA 18: Team Lead provides firing command for the wrong target (H5)
[107] (B) Scenario: The targeting system highlighted the correct target. However, the system switched targets while the Team Lead was maneuvering, and the Team Lead did not notice the switch.
[108] Same as above

representation of a different target. As a result, the UAS fixes a different target than the TL intends to fire on.[109]
        - *Lateral Coordination (Communication)*: Same as S-37.1.3, leading to the controllers acting on different tasks.
        - *Lateral Coordination (Observation)*: Same as S-4.4.3.
        - *Prediction*: Based on past experience, both controllers have high confidence in the ability of their collaborator to provide an adequate <fix, fire> command. As a result, they devalue the necessity to scrutinize their coordinated information to ensure the targets are consistent. This may be to the point in which they notice the wrong target designation, but reject the information as incorrect given their high confidence.
    - ***Decision-Making***:
        - There is no logic to rectify the lack of consensus on a specific target. As long as a set of other parameters are met (e.g., time window), the distributed decision-making does not check, attempt to enforce or rectify a lack of consensus on a specific target.
    - ***Capacity***:
        - One or more of the controllers are operating at a workload too high to verify the consistency of the target solution between the fix and fire task.
- ***Dynamic Membership:***
    - The set of targets is dynamic. As a result, TL providing fire and the UAS providing the fix confuse targets and focus on different ones. This may be due to how the software indexes the dynamic set of targets if a new target is labeled with the index of an old target that is no longer part of the mission.[110]
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** No new scenarios conceived.

S-4.4.9 (Step 4: Other Factors): Controllers on the team fix and fire onto different targets due to factors beyond those explored in interactions between the controllers on the team.

- No new scenarios.

Abstraction 2a – Type 3-4: Commands start/end too early/late relative to one another

The following UCCAs explore how control actions provided by the team are started or ended too early or too late relative to one another. Causal scenarios associated with the temporal sequencing of control actions consider how control actions and collaborative interactions internal to the team contribute to the UCCA. The same top-level scenarios employed in Abstraction 2b guide the analysis below.

---

[109] (B) Scenario: the command only contains an approximate location of the target, but there are several potential targets within range and the FMS selects wrong one, and command is fixed on a different target.
[110] (B) Scenarios: There is a delay from an earlier fix on target command that the FMS acts on instead, so command is fixed on a different target.

UCCA 15.1 (abstracted UCCA): The Team ends the fix command before it starts the fire command when the target fired on must be fixed [H3, H5].

UCCA 15.1.1 (refined): UAS1 ends the fix command before TL starts the fire command.

UCCA 15.1.3 (refined): UAS1 ends the fix command before UAS2 starts the fire command.

In Table A2-4, UCCA 15.1.2 has the same outcome but involves the same UAS ending the fix before it starts to fire, which is not a collaborative control problem.

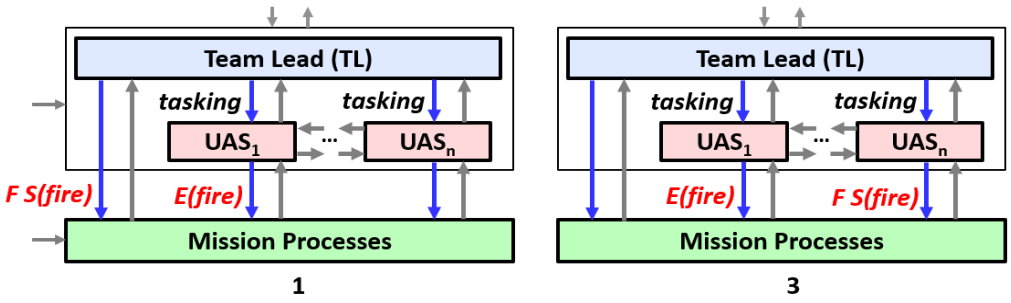


Figure A3-23. Control structures showing team ending the fix before it starts to fire

S-15.1.6 (Step 1: Top-Level Scenario 6): TL directs the UAS in a way that leads to unsafe temporal sequencing. Here, TL tasks a UAS to end fix too soon and another to start fire too late relative to each other.

- ***(Step 2: Internal Control) Unsafe Control Input:***
  - TL interprets direction from higher authorities that any fix must be ended by a certain time and/or that no fire may be provided before a certain time.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: TL believes (1) the fix command will end later than it does, (2) the fire command will start earlier than it does. ***Refinement***:
  - *Missing Feedback.* TL does not receive any feedback from UAS associated with uncertainty in meeting task timelines as specified by the TL. This leads TL to believe s/he can specify limited overlaps in the fix-fire tasks, which are not realistically achievable.[111]
  - *Missing Feedback.* TL interface does not display how the fix and fire commands align temporally as tasked relative to one another.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL has accurate information about the tasks as specified having an inadequate temporal sequence, but still chooses to provide those commands as such. ***Refinement***:
  - TL misunderstands how the UAS operate. S/he incorrectly believes that the UAS will adjust the timelines specified in the tasking so that the sequencing is adequate. As such, s/he does not actively correct the tasking sequence.
  - TL misunderstands how TL or UAS weapons systems operate. S/he incorrectly believes that if there is no fix provided, the weapon system will not fire (or vice

[111] (B) Scenario: Team Lead has an inaccurate mental model of how long it takes the AuC to implement a command upon receiving it.

versa) (Same as S-3.3.1). This reduces her/his perceived consequence in misspecifying the taskings.
    - TL misunderstands how the system operates, and s/he believes that a UAS has to be providing a fix before they can task a UAS to fire. As a result, TL delays tasking the fire until it is late to execute with respect to the fix.
- ***(Step 2: Internal Control) Unsafe Control Path:***
    - The interface pre-populates timelines associated with each task to reduce the operator workload. However, those timelines are inadequate in sequence.[112]
    - The tasking interface sends a time that is different from the one specified by the TL. This could be due to some default times preprogrammed that are substituted in under certain circumstances.[113]
    - The interface and messages as designed are prone to common timing errors (e.g., 12 vs 24hr, Local vs Zulu, international date line, changing time zones).[114]
    - One of the two taskings is delayed due to inadequate communication channels (see S-37.1.1). Thus, the <fire, fix> task is not received by the designated controller until after the controller that provided the coupled <fix, fire> task has acted.
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops****: Not applicable. Top-level scenario focuses on a TL internal control loop that is not closed through any other controller.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** *Not applicable. Top-level scenario focuses on a control decision from the TL only.*
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
    - One of the UAS tasked in the fix-fire coupled command is removed from the team. TL retasks another UAS in response but does not change the original tasking to compensate for any effects this change of controller has on the timeline.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S-15.1.7 (Step 1: Top-Level Scenario 7): TL adequately directs the UAS, but the way in which the UAS execute the tasks leads to unsafe temporal sequencing. Here, despite the tasking, one UAS ends the fix early and the other UAS starts the fire command late relative to one another.[115]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Another controller (e.g., Ground Station) directly overrides one of the UAS and changes its task timeline in a way that is inconsistent with the temporal sequencing of other controllers on the team. (Similar to S-47.1.7)
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:***

---

[112] (B) Scenario: The Team Lead implements the default values for identification.
[113] (B) Scenario: The transmission becomes corrupted
[114] (B) Scenario: The AuC miscalculates the time by assuming the command is supposed to last for 15 minutes instead of lasting until 15:00 PM.
[115] (B) UCA 33: Autonomous Controller implements a task after the target is no longer necessary or is out of range (H3)

- The UAS tasked to fix does not intend to end its command so early, but its targeting system malfunctions and it must stop the task. [116]
- The <TL, UAS> tasked to fire does not intend to fire so late, but its weapon system malfunctions and delays the weapons release.[117]

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** See Figure A3-2.
  - ***Feedback about Controlled Process received from Collaborators*:**
    - The <TL, UAS> tasked to fire does not receive feedback from the UAS providing the fix that the fire command should not be provided. For example, the controller tasked to fire has already fired one successful shot. The UAS providing the fix observes the target getting destroyed, stops providing the fix, and attempts to provide fire feedback to the controller firing. However, that feedback is not adequately received or interpreted by that controller (for reasons that can be further refined).
    - The UAS tasked to fix does not receive feedback from the controller tasked to fire regarding the adequacy of the fix command. As a result, that UAS now believes there is no controller that intends to fire, and it stops providing its fix. However, the feedback channel between the controllers was just inadequate (can be refined). The other controller does proceed with its plans to fire, near the same time the fix is ended.
  - ***Feedback about Collaborator control actions received from Controlled Process*:** The controller tasked to fire receives incorrect feedback from the target that leads it to believe the UAS is providing a fix. ***Refinement***:
    - The UAS tasked to fix observes the target getting hit by a weapon and assumes that the controller tasked to fire has completed its task. As such the UAS ends its fix. However, the weapon that struck was either one of multiple weapons to be fired or was fired by a different controller. In either case, the controller tasked to fire now launches its weapon but after the fix has ended.
    - UAS1 (tasked to fix) announces it must end its fix earlier than coordinated and before <TL, UAS2> (tasked to fire) can fire. However, <TL, UAS2> receives feedback from a fix beyond the time UAS1 announced it would stop. This occurs due to delays in the UAS1 control path or because another UAS provides a spurious fix. As a result, <TL, UAS2> incorrectly believes UAS1 can now accommodate the coupled fix-fire task, and proceeds to fire as the fix ends.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
  - ***Construction*:**
    - The algorithms executed by different controllers are inconsistent with one another regarding how they handle specified timelines. Examples:

---

[116] (B) Scenario: MFC does not open the weapons bay, there is an electrical inference that causes the bay to unlock, or a missile is loose in the bay and pushed through the bay doors.

[117] (B) Scenario: same as above.

- One controller believes the specified start time is the time at which the command must start to be provided. Another controller believes the start time indicates the time at which the vehicle must start traveling to provide the command once at its destination.
- The controllers are configured to handle time according to different formats (Local, Zulu, GPS time, …).[118]

- ***Initialization***:
  - The controller models are inconsistently initialized regarding when the task needs to occur. For example, the task is specified as "start in XX seconds (from time of receipt)". However, the time of receipt of the task may vary between controllers given how the message travels from node to node from the TL to the intended recipient along the network topology.
- ***Model Updates*:**
  - *Vertical Coordination (Control)*: See factors listed for S-15.1.6.
  - *Lateral Coordination (Communication)*:
    - One of the controllers attempts to notify the other controller that it will not be able to meet the specified timeline. For example, the controller tasked to fire wants to inform the UAS providing the fire that it will be late. However, the communications channel between the controllers is degraded the time update is not shared.
    - The controllers use different refences of time to coordinate timelines. One controller uses time since an event, whereas another controller uses absolute time. (Human-Machine) this may be aggravated due to asymmetry between humans and machines in semantic meaning and timing.[119]
    - Messages exchanged between controllers do not contain explicit information on the planned timeline. They are instead relative to past information, which may be misinterpreted, and continue to reinforce inconsistent models of time. For example, controllers exchange messages such as "no change in estimated time of arrival" instead of "current estimated time of arrival is X time".
  - *Lateral Coordination (Observation)*: The observed behavior of a teammate leads a controller to incorrectly assume that task execution will not have temporal sequencing issues.
    - For example, UAS1 (tasked to fix) observes <UAS2, TL> traveling to the fire command destination at a rate that will make it arrive before UAS1 stops fixing. However, it does not anticipate that <UAS2, TL> will slow down as it approaches the destination.

---

[118] (B) Scenario: The AuC miscalculates the time by assuming the command is supposed to last for 15 minutes instead of lasting until 15:00 PM.

[119] (B) Scenario: The AuC miscalculates the time by assuming the command is supposed to last for 15 minutes instead of lasting until 15:00 PM.

- Similarly, <UAS2, TL> (tasked to fire) observes UAS1 loitering in a way that is consistent with the intent to remain on station to provide a fix for an extended period of time. It incorrectly assumes it will not be problematic if <UAS2, TL> at the end of the time window. It does not know that UAS1 is loitering to conserve fuel and is struggling to meet the original timeline.
- One of the controllers assumes that it does not need to actively coordinate a delay with a peer because the peer will be able to observe the controller and anticipate the problem. For example, <UAS2, TL> tasked to fire is no longer able to make the time window coordinated for the other controller (UAS1) to provide the fix. <UAS2, TL> incorrectly believes that UAS1 will observe its progress, anticipate when it will actually arrive, and automatically adjust the fix window accordingly.

▪ *Prediction*: The controllers make assumptions about each other's timelines based on the type of task provided, but without explicitly knowing how they were tasked.
- For example, UAS2 tasked to fire assumes that UAS1 (tasked to fix) will provide the fix for X minutes, as that is the default period of time of a fix. However, UAS2 is not aware or does not consider that the tasking actually specified Y minutes of fix (where X>Y).

o ***Decision-Making*:**
▪ The distributed decision-making process is too slow for the dynamics of the joint-control problem. The controllers are unable to reach consensus on a new timeline before a previous and outdated timeline is executed.
- For example, UAS1 (tasked to fix) reduces its estimate for how long it can provide the fix, and starts a process with UAS2 (tasked to fire) to update the coordinated timeline. The distributed algorithm does not reach consensus until after UAS1 has ended the fix and UAS2 has fired late.

▪ By trying to over-optimize the usage of time by minimizing unnecessary time overlaps, the decision-making process is unable to produce a feasible solution.
- For example, UAS1 (tasked to fix) assesses that the planned overlap for the fix and fire commands is excessive. Thus, it plans to end the fix earlier so that it is available to execute other tasks. However, this pushes UAS2 (tasked to fire) to move up its start time for the fire command to maintain the same margin of overlap. This process repeats until UAS1 plans to end the fix earlier than UAS2 can start its fire command.

▪ The timeline coordinated between the UAS does not account for uncertainty in their ability to meet timing constraints. For example, the UAS tasked to fix communicates that it can provide a fix until Time1,

which it determines from its fuel quantity. However, its fuel consumption is then higher than planned and it now must end the task early.[120]

- ***Capacity***:
  - One of the controllers does not have the capability to assess timelines. For example, a UAS tasked to fire is not capable of computing an estimated time of arrival at the task given environmental influences on the controller (e.g., wind, obstacles, traffic disturbances). As a result, the controller takes on tasks with unrealistic time constraints that it is not able to meet.[121]

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-37.1.3. **In addition**:
  - The controller tasked to fix is removed from the team and its task is reassigned. However, the controller tasked to provide the coupled fire command is not aware of the retasking and therefore does not coordinate timeline updates as necessary.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***
  - Variations in the network topology introduce variations in the time of receipt of messages. If that time of receipt is used as a reference, that introduces variations in the perceived timelines for the different controllers, which can then lead to unsafe temporal sequencing. (Related to Initialization above)

S-15.1.8 (Step 1: Top-Level Scenario 8): TL control actions to the shared process are unsafe in temporal sequencing with how it directs the UAS. Here, despite how s/he tasked the UAS to fix, the TL starts to fire after the UAS ends the fix.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-15.1.6.
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-15.1.6. ***Refinement***:
  - *Incorrect Feedback*: TL receives inadequate feedback from the system leading her/him to believe s/he will reach the firing location <earlier, later> than s/he actually will. The estimate does not account for environmental disturbances which influence travel time.[122]
  - *Missing Feedback:* The updated time estimate information is not readily accessible to influence the task execution behavior of the TL. (Similar to above)
  - *Missing Feedback*. TL interface does not display how the fix performed by the UAS and the ability for the TL to navigate to the fire location align temporally. As such, the TL is unaware that s/he will not make the time window. (Similar to S-15.1.6)
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** Same as S-15.1.6. **In addition**:
  - TL is currently busy and prioritizes other operating tasks, but intends to update the timeline for the UAS to provide the fix to ensure adequate temporal sequencing. However, s/he later forgets to do this. (Similar to S-37.1.1)
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-15.1.7.

---

[120] (B) Scenario: The AuC algorithm miscalculates which UAV is optimal for performing a maneuver because, the algorithm does not account for environmental factors impacting the UAV's sensors.
[121] (B) Scenario: The AuC algorithm miscalculates which UAV is optimal for performing a maneuver because, the algorithm does not account for environmental factors impacting the UAV's sensors.
[122] (B) Scenario: The AuC algorithm miscalculates which UAV is optimal for performing a maneuver because, the algorithm does not account for environmental factors impacting the UAV's sensors.

- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-15.1.7.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** Same as S-15.1.7.
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** *No new scenario conceived.*
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S-15.1.9 (Step 4: Other Factors): Controllers on the team end the fix before starting to fire due to factors beyond those explored in interactions between the controllers on the team. ***Refinement***:

- ***Unsafe Process Behavior:***
  - The target employs counter-measures that make the controller tasked to fire believe that the fix command is being provided, even though it is not.
  - The target employs counter-measures that make the controller tasked to fix believe that the target has been fired on and destroyed.

UCCA 17.2 (abstracted UCCA): The Team starts the fire command before it starts the fix command when the target fired on must be fixed [H3, H5].

UCCA 17.2.1 (refined): TL starts the fire command before UAS1 starts the fix command.

UCCA 17.2.3 (refined): UAS1 starts the fire command before UAS2 starts the fix.

Another lower priority UCCA listed in Table A2-4 leads to the same UAS ending the fix before starting to fire, which is not a collaborative control problem.

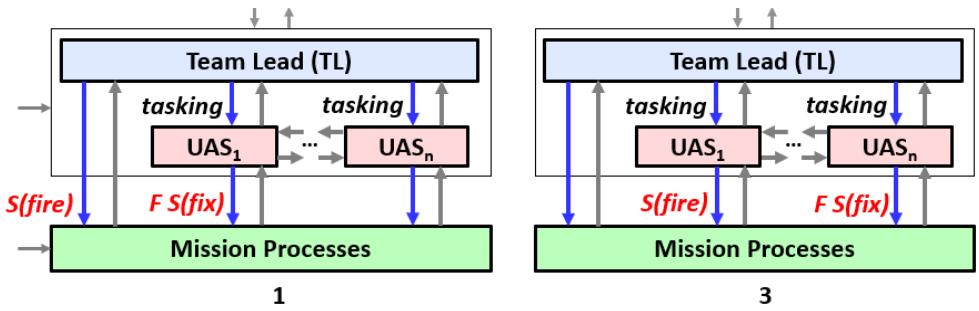


Figure A3-24. Control structures showing team starting the fire before it starts to fix

S-17.2.6 (Step 1: Top-Level Scenario 6): TL directs the UAS in a way that leads to unsafe temporal sequencing. Here, TL tasks a UAS to start fire to soon and another to start fix too late relative to each other.[123]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-15.1.6, but in reverse: fire cannot be provided after a certain time, fix cannot start before a certain time.
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: TL believes (1) the fix will start earlier than it does, (2) the fire will start later than it does. ***Refinement***: Same as S-15.1.6
- ***(Step 2: Internal Control) Inadequate Control Algorithm, Unsafe Control Path:*** Same as S-15.1.6.

[123] (B) UCA 22: Team Lead provides fire command before the right target has been targeted (H5)

- ***(Step 3: Collaborative Dynamic) Dynamic Membership, Dynamic Connectivity:*** Same as S-15.1.6.

S-17.2.7 (Step 1: Top-Level Scenario 7): TL adequately directs the UAS, but the way in which the UAS execute the tasks leads to unsafe temporal sequencing. Here, despite the tasking, (1) one UAS starts the fix late relative to (2) the other UAS starts the fire command early.[124]

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-15.1.7.
- ***(Step 2: Internal Control) Inadequate Process Model:*** See cognitive alignment.
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** See cognitive alignment.
- ***(Step 2: Internal Control) Unsafe Control Path:***
    - The UAS tasked to fix does not intend to start its command so late, but its targeting system malfunctions, and it cannot start on time. (Similar to S-15.1.7)
    - The UAS tasked to fire does not intend to fire so early, but its weapon system malfunctions and releases the weapon ahead of schedule. (Similar to S-15.1.7)
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** See Figure A3-2.
    - ***Feedback about Controlled Process received from Collaborators*:** <TL, UAS2> (tasked to fire) does not receive feedback from UAS1 providing the fix that the fire command should not be provided. (Same as S-15.1.7)
    - ***Feedback about Collaborator control actions received from Controlled Process*:** UAS1 (tasked to fix) announces it must start to fix later than coordinated and after <TL, UAS2> (tasked to fire) plans to fire. However, the <TL, UAS2> receives feedback from a fix ahead of the time UAS1 announced it would start. This occurs due to another UAS providing a spurious fix, or malfunction in the feedback path. <TL, UAS2> interprets this as a valid fix, even though it is not. (Similar to S-15.1.7)
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment, Dynamic Membership, Dynamic Connectivity:*** Same as S-15.1.7

S-17.2.8 (Step 1: Top-Level Scenario 8): TL control actions to the shared process are unsafe in temporal sequencing with how it directs the UAS. Here, despite how s/he tasked the UAS to fix, TL starts to fire before the UAS starts the fix.[125]

- ***All factors:*** Same as S-15.1.7.

S-17.2.9 (Step 4: Other Factors): Controllers on the team end the fix before starting to fire due to factors beyond those explored in interactions between the controllers on the team.

- ***All factors:*** Same as S-15.1.9.

[124] (B) UCA 37: Autonomous Controller releases a missile before receiving a command from the TL (H5)
[125] (B) UCA 22: Team Lead provides fire command before the right target has been targeted (H5)

Appendix 4: MUM-T Case Study – Dynamic Hierarchy Causal Scenarios

This short example illustrates how to identify causal scenarios for a system that exhibits dynamic hierarchy. Dynamic hierarchy is the one collaborative control dynamic defined in Chapter 3 that is not present in the Manned-Unmanned Teaming (MUM-T) baseline case study analyzed in Appendix 3. For this reason, the interactions exhibited by the MUM-T system are expanded here to include this type of interaction and demonstrate how it is handled in the causal scenario development process. This example is representative of the type of analysis that could be performed to explore the system safety considerations of a MUM-T concept alternative that includes dynamic hierarchy.

As described in Chapter 5.6, the following assumption is modified in the MUM-T system. The TL still tasks the UAS team as a whole to execute the search task, and delegates to the UAS team the authority to allocate that task to one of the controllers. However, now the UAS team considers the TL as one of the controllers to which it can assign the task. This creates dynamic hierarchy, in which the TL at a high level of work oversees the UAS team, but at a lower level the UAS team can provide a control action to the TL to go search.

The process to identify and refine UCCAs is unchanged with this dynamic, and as such the list of UCCAs is no different than the one developed in the main case study analysis. The modification to the system only impacts the UCCAs that involve the search command, which are consolidated into Table 5-18.

In the fourth step of the analysis, the causal scenarios are developed using the same process defined in Chapter 4.3. They are initialized using *top-level scenarios,* which describe the different possible control actions internal to the team that are relevant to the unsafe collective output. Dynamic hierarchy reflects internal control, so this is where it is addressed in STPA-Teaming. The addition of a new internal control action from the UAS to the TL generates new top-level scenarios. The scenarios are then refined using the same process as in the Appendix 3 analysis.

Two examples follow to show the development of scenarios related to dynamic hierarchy. They include scenarios for one of the Type 1-2 UCCAs and those for one of the Type 3-4 UCCAs. The other UCCAs listed in

Table 5-18 are not analyzed as part of the scope of this work, but their scenarios could be developed using the same method. The causal scenarios related to dynamic hierarchy are indexed as S_{DH} to differentiate them from those developed in the main case study in the previous appendix.

Type 1-2 UCCA Dynamic Hierarchy Example Scenarios

The scenario identification process for dynamic hierarchy uses the same general top-level scenarios as in the previous analysis. However, in each case, the top-level scenario must also consider how the UAS can issue control actions to the TL. The following describes how the top-level scenarios include this consideration and highlights the new cases.

1. Tasks Not Provided (Unsafe): A controller on the team does not provide tasks to other controllers on the team that are necessary for the team to execute safe collective control of the shared process. This includes:
    a. TL does not task all the UAS as necessary
    b. UAS team does not task TL to search as necessary **[NEW]**
2. Tasks Provided (Unsafe): A controller on the team provides tasks to other controllers on the team in a way that leads to unsafe collective control. This includes:
    a. TL provides tasks to multiple UAS that conflict with one another
    b. TL provides incorrect task(s) to the UAS
    c. TL provides task(s) to incorrect UAS
    d. UAS team provides incorrect search task(s) to TL **[NEW]**
    e. UAS team incorrectly tasks TL instead of allocating the task to a UAS **[NEW]**
3. Tasks Provided (Safe) but Not Executed as Tasked (Unsafe): A controller on the team provides task(s) to other controllers on the team adequately, but some of those controllers do not execute them properly, which leads to unsafe collective control. This includes:
    a. UAS do not execute some of the tasks provided by TL
    b. UAS execute incorrectly some of the tasks provided by TL
    c. Incorrect UAS execute some of the tasks provided by TL
    d. TL does not execute the search task provided by UAS team **[NEW]**
    e. TL executes incorrectly the search task provided by UAS team **[NEW]**
4. Tasks Not Provided (Safe) but Executed (Unsafe): Controllers on the team adequately do not provide certain task(s) to other controllers on the team, but some of those controllers execute them anyways, which leads to unsafe collective control.
    a. UAS execute tasks that were not tasked by TL
    b. TL executes a search task that was not tasked by UAS team **[NEW]**
5. Tasks and Tasker Control Actions Unsafe: A controller on the team provides control actions to the shared process that are unsafe in combination with otherwise adequate tasks provided by that controller to other controllers on the team. This includes:
    a. TL does not provide a control action that is necessary in combination with some of the tasks s/he provided to the UAS
    b. TL provides a control action incorrectly or in a way that conflicts with some of the tasks s/he provided to the UAS
    c. UAS team does not provide a control action that is necessary in combination with the search task it provides to TL **[NEW]**

UCCA 43.7 (abstracted UCCA): Controller C_i does not search and no other C_j searches when no targets have been found [H3]

UCCA 43.7.1 (refined): TL, UAS1, and UAS2 do not provide search command

The figure below shows how the Team Lead (TL) and the UAS can task each other to search given the dynamic hierarchy. Each case is traced to its relevant top-level scenario(s). The third case is not logical in the context of the UCCA.

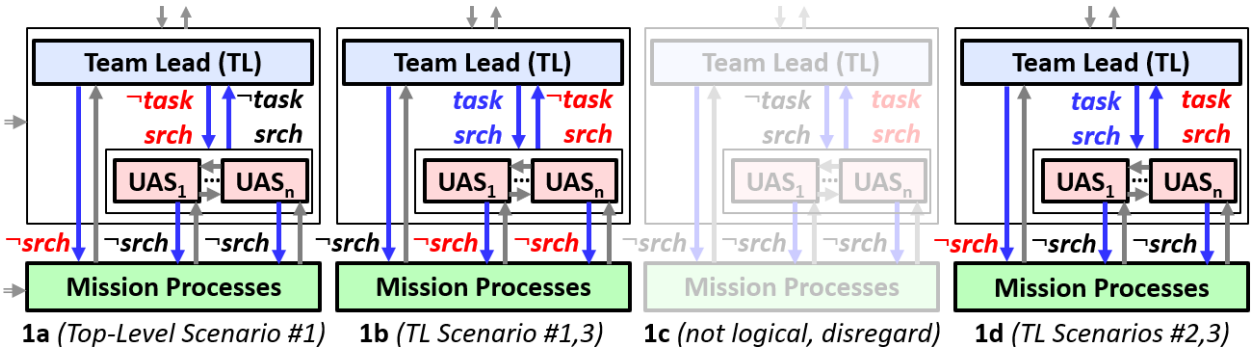


Figure A4-1. Internal Control Combinations that Result in No Controller Providing the Search

S_{DH}-43.7.1 (Step 1: Top-Level Scenarios #1): A controller on the team does not provide tasks to other controllers on the team that are necessary for the team to execute safe collective control of the shared process. In the UCCA context:
(3) TL does not task UAS team to search (and does not search her/himself) (see old S-43.7.1)
(4) UAS team does not retask TL to search (and does not allocate a UAS to search) **[NEW]**
The following focuses on item 2 and new scenarios

- ***(Step 2: Internal Control) Unsafe Control Input:*** UAS team interprets direction from another controller (e.g., Ground Station, another TL, a cyber attacker) that it should never retask the search to the TL, even if the TL is the only capable controller. **[NEW]**
- ***(Step 2: Internal Control) Inadequate Process Model:*** UAS Team has inadequate process model variables: it believes (1) it does not have authority to task TL, (2) that TL is intending to search, or (3) TL cannot search. **[NEW]**
    - ***Misinterpreted Information***: UAS Team receives information from the TL that it interprets as not being capable of doing the search (e.g., search sensor inoperative). **[NEW]**
    - ***Delayed Information***: TL recently changed status from unavailable to available to task. However, that information is slow to be processed by the UAS Team and the option to retask the search to the TL is not considered in time. **[NEW]**
- ***(Step 2: Internal Control) Inadequate Control Algorithm:***
    - TL does not want to be directed by machines, as s/he must make decisions for the team at a higher level than the UAS. Therefore, TL purposefully blocks her/his schedule so that the UAS never consider retasking the search back to the TL, even when the search is the highest priority and can only be fulfilled by the TL. **[NEW]**
    - UAS are programmed to assume that the TL will search if a UAS is not allocated (instead of actively retasking TL to search). **[NEW]**
- ***(Step 2: Internal Control) Unsafe Control Path:*** UAS Team intends to retask TL, but is unable to do so. (Similar to old S-37.1.1) ***Refinement***:
    - The TL interface does not emphasize when and how the UAS have retasked the search to the TL. This is aggravated by high TL workload. (Similar to old S-37.1.1)
    - TL does not want to be bothered by commands from the UAS Team and blocks the communication path to receive that command. The UAS Team continue to send the retask command without being aware the TL is ignoring them. **[NEW]**
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Does not apply.*

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
    - ***Construction***: The process models and/or control algorithms of the UAS are not adequately or consistently built across the team to support collaborative control. ***Refinement***:
        - The control algorithms on the UAS are not compatible with how the TL specifies the search task. This prevents the UAS from initializing or evaluating the option that the TL should be retasked to search. (Similar to old S-43.7.3)
        - The control algorithms of some of the UAS are configured with a software version that does not allow the UAS to retask anything to the TL. Similarly, the TL may have been told that the UAS cannot provide retasks, and therefore will ignore all such commands. **[NEW]**
    - ***Initialization***: Some of the controllers' process models are not adequately or consistently initialized across the team. ***Refinement***:
        - The UAS receive different information regarding the availability of the TL. This could occur as TL periodically updates her/his plans given slight changes in her/his model of the mission. The UAS are updated across different exchanges. As a result, they do not reach consensus that the TL can be retasked to search. **[NEW]**
        - The UAS have different beliefs about the current roles and responsibilities of other controllers on the team. For example, one UAS incorrectly believes that the TL has taken on the search task based on stale information. It disseminates this to the rest of the team and leads to the control algorithm to not retask the search to the TL. (Same as old S-43.7.3)
        - The way in which the retask is communicated to the TL leads her/him to believe another controller has been tasked. For example, the autonomy retasks the search to controller identifier (ID) #1, which the TL does not recognize as her/his own but is the autonomy's ID for the TL. **[NEW]**
    - ***Model Updates***: Some of the UAS's process models are not adequately or consistently updated. ***Refinement***:
        - *Vertical Coordination (Control):* The UAS Team attempts to retask the TL to perform the search, but the retasks are inconsistent with one another (e.g., some call for a retask whereas some do not, they call for different time windows, …) and the TL receives different versions of the coordinated output over time. Because of these inconsistencies, the TL (or her/his interface) rejects the retask as invalid. **[NEW]**
        - *Lateral Coordination (Communication):*
            - The information the UAS use to coordinate a solution regarding which controller to task to search is inconsistent. For example, the UAS receive state estimates from each other and the TL, which become slightly outdated due to small communications and processing delays. For planning, they then compare these

asynchronously outdated state estimates with their current own and derive different solutions for which controller should search. Due to lack of consensus, they do not retask it to the TL even though s/he is the best suited for it. (Similar to old S-37.1.3)

- UAS2 is temporarily disconnected and its model of the search task diverges from that of the team. UAS2 reconnects, contributes its divergent variables, disturbs team consensus, and leads the team to not retask the search to the TL. (Similar to old S-37.1.3)
- The information shared by the TL to the UAS Team is inadequate for the team to formulate a model of whether the TL is available for tasking (e.g., inadequate channels, semantic mismatch, etc…). Lacking this information, the team defaults to treating the TL as unavailable for the search task. **[NEW]**

▪ *Lateral Coordination (Observation):* The UAS observe TL maneuvering in a way consistent with executing other tasks and therefore assess that the TL is not available to execute the search. (Similar to old S-37.1.3)

▪ *Prediction:*

- A UAS shares that it has been tasked by the TL to provide a fix, but no additional information is available regarding which controller will execute the coupled fire task. As such, the UAS predict that the TL will be the controller that takes on that task and is, therefore, unavailable to be tasked to search. **[NEW]**
- The last several times the UAS team retasked the TL to search, the TL did not execute it. As a result, some of the UAS update their models regarding their ability to retask the TL and veto future plans that involve this option. **[NEW]**

o ***Decision Making***: The process UAS use to decide what actions they provide is inadequate or inconsistent across the team. ***Refinement***:

▪ Parameters in the distributed decision-making algorithm bias the decision-making against retasking the search to the TL. It is nearly impossible to reach consensus on this type of solution. **[NEW]**

▪ It takes too long for the UAS to reach consensus on whether or not to retask the TL onto the search. Before the decision can be made, TL loses patience and re-issues the search command, which reinitiates the planning process. This cycle repeats itself until the task is no longer relevant. **[NEW]**

o ***Capacity***: *No new scenario conceived*

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Changes in the set of UAS that participate in the team contribute to this UCCA. ***Refinement***:
  - The team must reach a majority or unanimity among the UAS to retask the search to the TL. However, the set of controllers fluctuates too much to achieve this threshold, so the team is unable to retask. **[NEW]**
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:***

- The UAS Team has reached consensus on retasking the search. As the solution is broadcast to the TL, another UAS that was disconnected from the team connects and shares its default plan that does not task anything to the TL. This causes this assignment to be dropped (either revoked by UAS Team or rejected by TL). **[NEW]**

S_{DH}-43.7.2 (Step 1: Top-Level Scenarios #2): A controller on the team provides tasks to other controllers on the team in a way that leads to unsafe collective control. Here, the UAS team incorrectly retasks the TL, who cannot search, instead of allocating the task to a UAS. [DH]

- ***(Step 2: Internal Control) Unsafe Control Input:*** The UAS team receives direction from another controller (e.g., Ground Station, another TL, a cyber attacker) that it should favor retasking the search command to the TL. (Similar to S_{DH}-43.7.1)
- ***(Step 2: Internal Control) Inadequate Process Model:*** The UAS Team has the following inadequate process model variables: (1) TL is capable and available to search, (2) TL is the controller best suited to search. ***Refinement***:
  - *Missing Information*: UAS Team does not receive information from TL regarding its inability to search (e.g., search sensor is inoperative). (Same as S_{DH}-43.7.1)
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** see Cognitive Alignment Decision-Making.
- ***(Step 2: Internal Control) Unsafe Control Path:***
  - UAS Team intends to assign a UAS to the search, but unintentionally provides what the TL interprets as a retask to her/him. For example, the UAS Team broadcasts that UAS ID #XX is selected for the search task, which is similar to that of the TL. TL sees the assignment, incorrectly assumes it is meant for her/him, and cancels the task altogether believing that no UAS will take it on. **[NEW]**
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Not applicable.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The controllers on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
  - ***Construction*:** The controllers on the team are not all consistently configured to evaluate the availability of the TL to take on a retask. TL provides the search task in a way that s/he believes indicates her/his inability to do it. Some of the UAS treat this information differently and push to reach consensus that the TL is the controller best suited for the task. This could arise due to problems in configuration or training of both the human and the machines. **[NEW]**
  - ***Initialization***: UAS receive different versions of information regarding the TL availability. TL periodically updates her/his availability given slight changes in her/his model of the mission, and the UAS receive this information across different sharing cycles. Thus, slightly outdated information is considered in incorrectly reaching consensus that the TL can be retasked. (Similar to S_{DH}-43.7.1)
  - ***Model Updates***: Some of the UAS's process models are not adequately or consistently updated. ***Refinement***:
    - *Vertical Coordination (Control): No new scenario conceived.*
    - *Lateral Coordination (Communication):*

- UAS2 is temporarily disconnected and its model of the search task, or ability and availability of controllers on the team, diverges from the team. UAS2 reconnects and now contributes its divergent model variables. This disturbs team consensus into reaching the incorrect solution to retask the TL. (Similar to old S-37.1.3)
- The information shared by the TL is misinterpreted by the UAS as being available for a retask, even though the TL was indicating s/he is not (e.g., semantic or syntax mismatch in message composition). (Similar to S_{DH}-43.7.1)

▪ *Lateral Coordination (Observation):* The UAS observes the TL maneuvering in a way that is inconsistent with executing other tasks and therefore assess that the TL is available to execute the search. This contributes to the UAS team retasking the search to the TL. (Similar to old S-37.1.3)

▪ *Prediction:*

- The way in which the TL has tasked the UAS team to execute fix and fire tasks leads the UAS to predict that the TL intends to take on the search task. As such, the UAS team retasks the search to the TL even though s/he is unable to do so. (Similar to S_{DH}-43.7.1)
- The last several times the UAS team allocated the search to the TL, the TL overrode the decision and took on the search her/himself. As a result, some of the UAS update their models regarding their ability to retask and push for retasking the search to the TL in future plans. (Similar to S_{DH}-43.7.1)

o ***Decision Making***: Same as S-43.7.3, **in addition**:

▪ Parameters in the distributed decision-making algorithm bias retasking the search to the TL over other solutions. (Similar to S_{DH}-43.7.1)

▪ UAS Team cannot reach consensus over the intent of the search task. Rather than allocate an ill-defined task to one of the automated controllers, it retasks it to the TL as a method of rejecting the task so that the TL can reformulate it. However, the TL interprets the retask literally and either attempts to do it unsuccessfully or drops it altogether. **[NEW]**

o ***Capacity***: TL does not have the workload capacity to provide all the planning information required by the UAS to determine the TL availability for tasking. The mission is too dynamic and each change takes too much effort to input. In addition, human-machine asymmetry also requires additional effort to convey intent to the machines. As a result, the machines do not know the TL is unavailable and incorrectly task the TL to search. **[NEW]**

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:***
  - The set of tasks to control in the mission changes dynamically. It is not practical for the TL to continually update the UAS team on her/his intentions and availability. (Related to capacity above)

- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** The UAS team becomes disconnected into two (or more) subnetworks after the TL has provided the tasking. Each group performs its own decision-making on which controllers should be allocated for the search task. Two groups provide different plans back to the TL (e.g., one group retasks TL and the other allocates a UAS, or both groups retask but with differing parameters). As a result, the retask is rejected as invalid by the TL. **[NEW]**

S_{DH}-43.7.3 (Step 1: Top-Level Scenarios #3): A controller on the team provides adequate task(s) to other controllers on the team, but some of those controllers do not execute them as provided, which leads to unsafe collective control. In the UCCA context:
(1) TL tasks the UAS team to search and no UAS searches. (See old S-43.7.3)
(2) the UAS team retasks the search but the TL does not do it. **[NEW]**
The following focuses on item 2 and new scenarios.

- ***(Step 2: Internal Control) Unsafe Control Input:*** TL (mis)interprets direction from higher authorities that retasks from the UAS are compromised and should not be obeyed. **[NEW]**
- ***(Step 2: Internal Control) Inadequate Process Model:*** TL has the following inadequate process model variables: (1) UAS do not have the authority to retask the search to the TL, (2) the retask command from the UAS is not valid, and (3) no retask has been issued. **[NEW]** ***Refinement***:
    - *Missing Information:* Some UAS do not provide the required authentication to the TL (or her/his interface systems) to accept the retask command. **[NEW]**
    - *Delayed Information:* TL retrieves the retask command with so much delay (e.g., delays in info exchange, processing, interface display) that s/he questions if it is still valid. **[NEW]**
    - *Missing Information:* TL's own mental model of the search allocation expected the UAS would allocate the search to a UAS instead of retask it to the TL. The automation lacks transparency into how it arrived at that solution. TL does not trust it and therefore does not execute it. **[NEW]**
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL is aware of the retask command by the UAS and is confident in its validity, however, s/he chooses not to execute it. ***Refinement***:
    - (Human-Machine) TL refutes the concept that a machine has authority over her/him and exhibits a subconscious bias against following any retask commands. **[NEW]**
    - Based on experience, TL anticipates mission tasks will arise that s/he will need to deal with. As a result, s/he elects not to execute the retasked search command to avoid getting bogged down. **[NEW]**
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S-37.1.3
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Does not apply.*
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** The UAS on the team have inadequate cognitive alignment relative to one another. ***Refinement***:
    - ***Construction***: The process models and/or control algorithms of the UAS are not adequately or consistently built across the team to support collaborative control. ***Refinement***:

        - TL and the UAS are not compatible with how they both specify and interpret search tasks. A retask should be a reflection of the search task provided by the TL. However, task misinterpretation occurs when the UAS receive the initial tasking, and again when the TL receives the retask. The retask looks inconsistent with what the TL initially defined. This results in the TL losing confidence in the retask, or in her/him not knowing how to decode or the intent in the retasked instructions. **[NEW]**
    - ***Initialization***: Some of the UAS's process models are not adequately or consistently initialized across the team. ***Refinement***:
        - UAS receive different versions of the task specification by the TL. These variances lead to the UAS Team to rescope the search task (e.g., by averaging some of the different parameters), before retasking it to the TL. Upon receipt of the retask, the TL does not understand why the search task has been modified and therefore loses confidence that the UAS team allocated it correctly. S/he does not perform the search. **[NEW]**
        - The UAS Team has only been conveyed a subset of the overall mission tasks. Their automated planning does not consider some of the tasks the TL plans to address. As a result, the UAS Team retask is rejected by the TL even though it is the best possible plan the UAS could have generated using the information available to them. **[NEW]**
    - ***Model Updates*****:** Some of the UAS's process models are not adequately or consistently updated. ***Refinement***:
        - *Vertical Coordination (Control):* Does not apply.
        - *Lateral Coordination (Communication):* Same as S_{DH}-43.7.1
        - *Lateral Coordination (Observation):* TL observes the traffic flow between the UAS Team and finds it inconsistent with her/his past experience in how UAS allocate search commands. The TL lacks confidence that the retask is valid and does not take it on. **[NEW]**
        - *Prediction:* The solution of the UAS Team to retask the search is different from what the TL had predicted. Based on the state of the team, the TL had anticipated with high confidence that a specific UAS would be selected to search. Thus, TL rejects the validity of the retasked command. **[NEW]**
    - ***Decision Making*****:** Same as old S-43.7.3
        - The UAS are unable to reach consensus on a plan. A majority of them elect to retask the search to the TL, but others generate different solutions. The solutions are all provided to the TL, who does not know how to handle the disparate outputs, and thus does not execute the retasked search. **[NEW]**
        - Similar to above, the UAS are unable to reach consensus, and each one "bombards" the TL with their disparate version of the task allocation. The UAS reiterate these instructions cyclically to ensure the TL has the most up-to-date command. The TL is confused and does not execute the retasked search. **[NEW]**
    - ***Capacity***: No new scenario conceived

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S_{DH}-43.7.1
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Same as S_{DH}-43.7.1

S_{DH}-43.7.4 (Step 1: Top-Level Scenarios #4): A controller on the team adequately does not provide tasks to others on the team, but some of those controllers execute them anyways, which leads to unsafe collective control. *Not applicable in the UCCA context* (Same as S-43.7.4)

S_{DH}-43.7.5 (Step 1: Top-Level Scenarios #5): A controller on the team provides control actions to the shared process that are unsafe in combination with otherwise adequate tasks provided by that controller to other controllers on the team. In the UCCA context:
(3) TL does not search when s/he does not task the UAS team to search. (See old S-43.7.5)
(4) UAS team does not allocate a UAS when it does not retask the TL to search. (See old S-43.7.3)

S_{DH}-43.7.9 (Step 4: Other Factors): *Not specific to dynamic hierarchy* (see old S-43.7.9)

Type 3-4 UCCA Dynamic Hierarchy Example Scenarios

The following UCCAs explore how multiple controllers start or end control actions too early or too late relative to one another. As with the Type 1-2 UCCA above, the Type 3-4 UCCA use the same general top-level scenarios as in Appendix 3. However, those are expanded as follows to consider how the UAS may provide control actions to the TL given the dynamic hierarchy.

1. Tasks as Provided Lead to Unsafe Sequencing: A controller on the team provides tasks to other controllers on the team in a way that leads to unsafe sequencing. This includes:
    a. TL tasks UAS in a way that leads to unsafe temporal sequencing.
    b. UAS team tasks TL to search in a way that leads to unsafe temporal sequencing. **[NEW]**
2. Tasks Provided (Safe) but Executed in Unsafe Sequencing: A controller on the team provides adequate task(s) to other controllers on the team, but the way in which those controllers execute the tasks leads to unsafe temporal sequencing. This includes:
    a. TL adequately tasks the UAS, but the UAS execute those tasks in a way that leads to unsafe temporal sequencing.
    b. UAS team adequately tasks the TL to search, but TL executes the search in a way that leads to unsafe temporal sequencing. **[NEW]**
3. Tasks and Tasker Control Actions are Unsafe in Sequencing: A controller on the team provides control actions to the shared process that are unsafe in temporal sequencing in combination with otherwise adequate tasks provided to other controllers on the team. This includes:
    a. TL control actions to the process are unsafe in temporal sequencing with other adequate tasks s/he provided to the UAS.
    b. UAS Team control actions to the process are unsafe in temporal sequencing with the adequate search tasks it provided to the TL. **[NEW]**

UCCA 56.3: Controller Cj ends providing search command before Ci starts providing search command when that creates an excessive gap in a search handoff [H3]

UCCA 56.3.1: TL ends providing search before UAS1 starts providing search.

UCCA 56.3.2: UAS1 ends providing search before TL starts providing search.

UCCA 56.3.3: UAS2 ends providing search before UAS1 starts providing search.

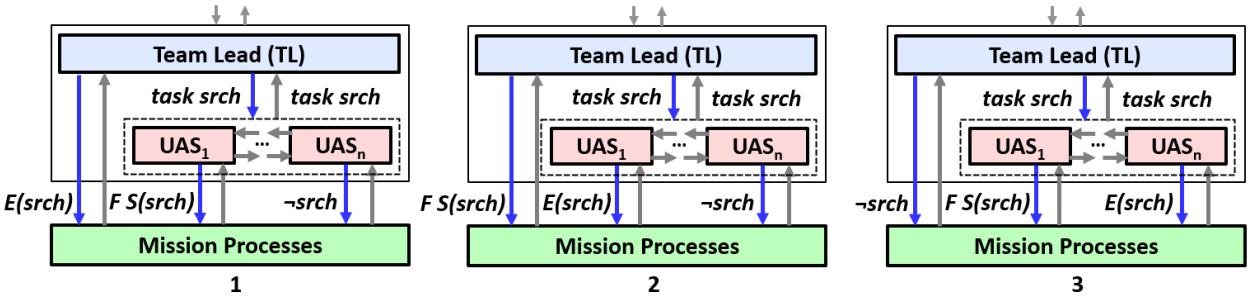


Figure A4-2. Control Structures for the Three Refined UCCAs

S_{DH}-56.3.6 (Step 1: Top-Level Scenarios #6): A controller on the team tasks other controllers on the team in a way that leads to unsafe sequencing. In the UCCA context:

(1) TL tasks the UAS to start or end the search in a way that creates a gap with the TL's planned start or end to the search (see old S-56.3.1)

(2) UAS team retasks TL to start/end search in a way that creates a gap with the UAS's planned start or end to the search **[NEW]**

The following focuses on item 2 and new scenarios.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-47.1.2
- ***(Step 2: Internal Control) Inadequate Process Model:*** UAS Team has the following inadequate process model variables: (1) the time windows retasked do not constitute an excessive gap, (2) TL is capable of fulfilling the retasked time window. This can be due to inadequate coordination information with the TL. ***Refinement***:
  - *Missing Feedback.* UAS Team does not know when TL can reach a certain location, nor how long s/he can perform the task. It is missing feedback regarding the TL aircraft performance (time to travel, remaining fuel endurance, …) as well as information regarding the other tasks intended to be fulfilled by the TL. **[NEW]**
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** UAS Team has accurate information about the search and the timing associated with controllers on the team, but still chooses to retask the TL to hand off the search with an excessive gap. ***Refinement***:
  - [Human-Machine] The automation has been designed to not excessively interrupt the human. The UAS change the search handoff window to adapt to changes in the mission. However, they do not retask the TL with the changes to avoid exceeding the designed human interruption rate. **[NEW]**
  - TL wants to minimize interruption by the machines and intentionally places a limit on how often the machines can interrupt her/him with changes in retasks. However, that does not prohibit the UAS from changing the handoff schedule for a UAS, even if the handoff is between a UAS and the TL. (Related to above)
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S_{DH}-43.7.1
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** *Not Applicable.*

- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
  - ***Construction***: Same as S-56.3.7
  - ***Initialization***: The UAS have inconsistent information regarding the TL's ability to reach a location or how long s/he can perform the search task for. This may be due to dynamic connectivity and certain UAS having stale information. As a result, the UAS begin their distributed planning of the handoff with different information.
  - ***Model Updates***: Elements of the process models are not adequately or consistently updated across the team. ***Refinement***:
    - *Vertical Coordination (Control):* The UAS Team over-controls the coordination of the handoff between a UAS and the TL by sending frequent fine-tuning updates. It has tight control authority over the UAS and can send frequent updates that are followed immediately. However, its control authority over the TL is looser, as information exchange is less frequent, and less precise due to human-machine asymmetry, and the TL has her/his own agenda which may not align with how the UAS Team is coordinating the handoff. The difference in control authority over the two types of controllers leads to the gap in handoff. **[NEW]**
    - *Lateral Coordination (Communication):* It is more efficient for the UAS to change UAS taskings than it is for TL taskings. Inadequate information flow between the UAS leads to lack of consensus on how to handle changes needed in the tasking of the TL for the handoff. **[NEW]**
      - Communication channels may be degraded due to jamming, fading, and equipment damage. (Similar to S-37.1.3)
    - *Lateral Coordination (Observation): No scenarios conceived.*
    - *Prediction: No scenarios conceived.*
  - ***Decision-Making***: There is asymmetry in the time needed to make decisions to reallocate a UAS and retask the TL (related to coordination by communication). It takes too much time for the UAS team to issue a decisive time window for the TL to take over the search. By the time the TL receives the task, it is not possible for her/him to proceed to the search area and take it over from a UAS without an excessive handoff gap. **[NEW]**
  - ***Capacity***: Same as S-37.1.3
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S_{DH}-43.7.1
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** Due to changes in connectivity between different UAS and the TL, the TL receives different versions of the retask and its updates at different times. A UAS which contributed to a previous version of the retask becomes disconnected before the remainder of the team updates the retask and provides it to the TL. The disconnected UAS later reconnects with the TL and shares the now stale version of the retask, which is the one the TL executes. **[NEW]**

S_{DH}-56.3.7 (Step 1: Top-Level Scenarios #7): A controller on the team adequately tasks other controllers on the team, but those controllers execute the tasks in a way that leads to unsafe temporal sequencing. In the UCCA context:

(1) A UAS ends its search before another UAS starts in a UAS-UAS handoff (see old S-56.3.1)
(2) The retasked TL starts/ends providing the searching too late/early in a way that leads to a gap in a TL-UAS handoff **[NEW]**

The following focuses on item 2 and new scenarios.

- ***(Step 2: Internal Control) Unsafe Control Input:*** The TL (mis)interprets direction from higher authorities that retasks from the UAS should be conducted earlier or later due to a flaw with UAS time keeping (e.g., GPS time vs UTC time offset). (Similar to S_{DH}-43.7.3)
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as old S-56.3.8
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** TL accepts the retask from the UAS to get resources moving, but never intends to fulfill that commitment. TL has ultimate authority over the team and does not feel obligated to fulfill commands provided by the machines. Her/his intent is to take charge closer to the handoff window to realign the handoff with her/his agenda. However, s/he becomes consumed in other tasks and forgets to do so at the required time. **[NEW]**
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as old S-47.1.2
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-56.3.8.
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:*** Same as S_{DH}-43.7.3
- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as old S-56.3.8.
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** *No new scenario conceived.*

S_{DH}-56.3.8 (Step 1: Top-Level Scenarios #8): A controller on the team provides control actions to the shared process that are unsafe in temporal sequencing with otherwise adequate tasks provided to other controllers on the team. In the UCCA context:
(3) TL executes her/his search in a time window inconsistent with the one s/he tasked to the UAS for a TL-UAS handoff. (See old S-56.3.8)
(4) UAS allocate a portion of the search to a UAS in a way that is inconsistent with how they retasked the other portion to the TL for a TL-UAS handoff. **[NEW]**

The following focuses on item 2 and new scenarios.

- ***(Step 2: Internal Control) Unsafe Control Input:*** Same as S-47.1.2
- ***(Step 2: Internal Control) Inadequate Process Model:*** Same as S-56.3.8
- ***(Step 2: Internal Control) Inadequate Control Algorithm:*** UAS Team has accurate information about the search and the timing associated with controllers on the team, but still chooses to allocate a UAS in a way that leads to an excessive gap. ***Refinement***:
    - The UAS assume there is flexibility in the schedule to retask the TL and that the TL will compensate for some changes from the planned handoff. The assumption may be pre-programmed or learned over time. The UAS team, therefore, reoptimizes to the allocation of the UAS in order to service other tasks. **[NEW]**
- ***(Step 2: Internal Control) Unsafe Control Path:*** Same as S_{DH}-56.3.7
- ***(Step 3: Collaborative Dynamic) Mutually Closing Control Loops:*** Same as S-56.3.8
- ***(Step 3: Collaborative Dynamic) Cognitive Alignment:***
    - ***Construction***: The control algorithms on the UAS are not compatible with one another to resolve a temporal sequencing issue between two different types of controllers (human TL and machine UAS). Some UAS bias solving the issue by

changing the TL's retasked time window. Others bias changing the UAS allocation. This creates conflict in the collective control output of the UAS. **[NEW]**

- ***Initialization***: Similar to S_{DH}-56.3.6, but the UAS have inconsistent information regarding the ability of the different UAS to perform the UAS time window of the search task.
- ***Model Updates*:** Same as S_{DH}-56.3.6
- ***Decision Making*:** The UAS Team tries to over-optimize the allocation of UAS to meet the time window for the handoff retasked to the TL. Repeated changes to the allocation lead to algorithmic churn that delays a UAS from decisively being assigned the task and start traveling to make the handoff. This is amplified if the automation amplifies the need to optimize timelines for handoffs with human controllers. (Similar to S-37.1.3)
- ***Capacity***: No scenarios conceived.

- ***(Step 3: Collaborative Dynamic) Dynamic Membership:*** Same as S-56.3.8
- ***(Step 3: Collaborative Dynamic) Dynamic Connectivity:*** No scenarios conceived.

S_{DH}-56.3.9: (Step 4: Other Factors): *Not specific to dynamic hierarchy* (see old S-56.3.9)

Appendix 5: MUM-T Case Study – Safety-Guided Requirements & Constraints

This appendix presents the safety constraints derived from the Manned-Unmanned Teaming (MUM-T) case study STPA-Teaming analysis. These items are organized according to the safety-guided design framework introduced in Chapter 6. The scope of the analysis is limited to Levels 1 and 2 of that framework.

The safety constraints at Level 2 are presented in a generalized form. They do not specify the controller names called out in the case study (e.g., TL, UAS_1, UAS_n) nor the control actions (e.g., fix, fire, search). Instead, they more generally refer to the types of controllers involved (e.g., human or machine), the hierarchal relationships of the controller (e.g., team leadership), and interdependencies between control actions (e.g., coupled actions like the fix and fire commands).

The general form keeps the set of safety constraints more concise. One general constraint may be specified to mitigate or eliminate the unsafe interactions between multiple different controllers and different control actions. As such, many of these constraints are traced to multiple scenarios. The general form also makes the results more broadly applicable to other collaborative control systems with similar control relationships. However, a systems engineering team could alternatively specify safety constraints using the actual controllers and control actions should those need to be specified for design purposes.

1. Level 1: MUM-T System-Level Safety Constraints

In Level 1 of the safety-guided design framework, the safety constraints are derived directly from the system-level hazards identified in Step 1 of STPA. These constraints inform, at a high level, the behaviors that must be enforced to prevent the system from reaching a hazardous state. They also specify how to prevent a loss from occurring should the system enter such a state. The safety constraints labeled with an (*) were obtained from the baseline STPA analysis performed by Robertson [53].

SC-1.1.1: Aircraft must satisfy minimum separation requirements from other aircraft and objects. [H1]*

SC-1.1.2: If aircraft violate minimum separation, then the violation must be detected, and measures must be taken to prevent a collision. [H1]*

SC-1.2.1: Aircraft control must not be lost given any possible input the flight controller may provide. [H2]*

SC-1.2.2: If aircraft control is lost, the loss of control must be detected, and measures must be taken to regain aircraft control. [H2]

SC-1.3.1: The system must accomplish planned mission operations. [H3]

SC-1.3.2: If the system does not accomplish planned mission operations, the lack of accomplishment must be detected, and measures must be taken to regain the ability to accomplish those operations. [H3]

SC-1.3.3: If the system performed unplanned operations that violate rules of engagement, the execution must be detected, and measures must be taken to stop those operations. [H3]

SC-1.4.1: Aircraft should not depart approved airspace [H4]*

SC-1.4.2: If aircraft violate approved airspace constraints, then the violation must be detected and measures taken to prevent encounters with the enemy. [H4]*

SC-1.5.1: The system must not fire at friendly assets or forces. [H5]*

SC-1.5.2: If the system fires at a friendly, then this condition must be detected and measures taken to prevent impact. [H5]

2. Level 2: MUM-T Safety Constraints Derived from STPA-Teaming

In Level 2 of the framework, the safety constraints are derived from the results of the hazard analysis performed on a conceptual architecture of the system. In this case, the safety constraints of the MUM-T system are derived from the Unsafe Combination of Control Actions (UCCAs) and the causal scenarios developed using STPA-Teaming in Appendices 2 and 3.

The analytical structure of STPA-Teaming links the UCCAs and scenarios to the different types of collaborative control dynamics implemented in the control structure. As such, the resulting safety constraints are also traceable to the decision to incorporate these types of interactions. The safety constraints presented in this appendix are organized according to the collaborative control dynamics they are primarily traceable to using the following labels.

CA: Cognitive Alignment	**DH**: Dynamic Hierarchy	**LC**: Lateral Coordination
DA: Dynamic Authority	**DM**: Dynamic Membership	**SA**: Shared Authority
DC: Dynamic Connectivity	**MC**: Mutually Closing Loops	**TA**: Transfer of Authority

The safety constraints derived from the UCCAs are traced to the three dynamics that relate to the authority of the controllers over the shared process. In this case study, *dynamic authority* applies primarily to Type 1-2 UCCAs found using Abstraction 2b (combination of controllers issuing a common control action). *Transfer of authority* is addressed in Abstraction 2b Type 3-4 UCCAs, as those focus on handoffs. Finally, *shared authority* is exhibited in all UCCAs, but it more uniquely influences Abstraction 2a UCCAs (combinations of different control actions provided by the team). The safety constraints for the scenarios developed under the UCCA carry the same traceability. This applies to the top-level scenarios and the refined scenarios below them.

In addition, the safety constraints derived from the refined scenarios are also traceable to the collaborative factors used in the refinement template shown in Figure 4-21. These include CA, LC, MC, DM, and DC from the list above. In the following analysis, those constraints are organized according to these five collaborative control dynamics. To simplify the presentation, the labels for DA, TA, and SA are not shown, but they are implied based on the UCCA they are derived from.

Because this research focuses on collaborative control, the scenarios that were refined from the traditional feedback control causal factors addressed in baseline STPA were not all analyzed to produce safety constraints. These are the scenarios associated with unsafe control input, inadequate process models, inadequate control algorithm, unsafe control path, unsafe feedback path, and unsafe process behaviors. A subset of these constraints is included at the end of this

appendix for demonstration purposes. As with other safety constraints, these items are traceable to DA, TA, SA based on the UCCA they are derived from.

Safety Constraints Primarily Traced to Dynamic Authority (DA)

The safety constraints below are derived from Abstraction 2b Type 1-2 UCCAs and their top-level scenarios. Their labels have the following convention. In SC-2.DA.2.1, "SC-2" means a safety constraint found in Level 2 of the safety guided framework. "DA" designates the collaborative control dynamic under which the constraint is organized. ".1" means it is derived from top-level scenario #1.

SC-2.DA.1: One of the controllers on the team must provide a control action if that action is necessary and the necessary resources to provide it are available. [UCCA 37.1, UCCA 40.4, UCCA 43.7, UCCA 1.1] [DA, SA]

SC-2.DA.1.1: The team leadership must direct its teammate(s) to provide the control action if it does not plan to provide it itself. [S-37.1.1, S-40.4.1, S-43.7.1, S-1.1.1] [DA, SA]

SC-2.DA.1.2: The team leadership must not direct multiple teammates to provide a control action if multiple controllers providing that action results in an unsafe conflict. [S-37.1.2, S-40.4.2, S-42.6.2, S-1.1.2, S-3.3.2] [DA, SA]

SC-2.DA.1.3: If a controller is properly directed by its team leadership to provide a control action, that controller must provide that action as directed. [S-37.1.3, S-40.4.3, S-1.1.3] [DA, SA]

SC-2.DA.1.3.1: If a set of controllers as a whole is properly directed by the team leadership to provide a control action, then that set must allocate a controller to provide the action. [S-43.7.3, S-1.1.3] [DA, SA]

SC-2.DA.1.4: N/A. Top-Level Scenario 4 does not apply to the UCCAs.

SC-2.DA.1.5: The team leadership must provide a control action itself if it is capable, that action is necessary, and it does not direct its teammate(s) to provide the action. [S-40.4.5, S-43.7.5, S-1.1.5] [DA, SA]

SC-2.DA.2: A controller that is not capable or available to provide a control action should not provide it if another controller is capable and available. [UCCA 38.2, UCCA 41.5] [DA, SA]

SC-2.DA.2.1: N/A. Top-Level Scenario 1 does not apply to the UCCAs.

SC-2.DA.2.2: The team leadership must not direct a controller to provide a control action that is not capable or available to do instead of directing a controller that is. [S-38.2.2, S-41.5.2, S-1.1.2] [DA, SA]

SC-2.DA.2.3: If the team leadership directs a capable and available controller to provide a control action, then no other controller must provide that action. [S-38.2.3, S-41.5.3, S-39.3.3, S-42.6.3] [DA, SA]

SC-2.DA.2.4: A controller must not provide control action u_i unless it is directed by the team leadership. [S-41.5.4, S-42.6.4] [DA, SA]

SC-2.DA.2.5: The team leadership must not provide a control action for which it is not capable or available if it can direct a capable and available teammate to provide it. [S-41.5.5] [DA, SA]

SC-2.DA.3: Multiple controllers must not provide the same control action if that creates an unsafe conflict. [UCCA 39.3, UCCA 42.6] [DA, SA]

SC-2.DA.3.1: N/A. Top-Level Scenario 1 does not apply to the UCCAs.

SC-2.DA.3.2: See SC-2.DA.1.2

SC-2.DA.3.3: See SC-2.DA.2.3

SC-2.DA.3.4: See SC-2.DA.2.4

SC-2.DA.3.5: The team leadership must not provide control action u_i and direct a teammate to provide u_i if multiple controllers providing u_i results in an unsafe conflict. [S-42.6.5] [DA, SA]

SC-2.DA.4: A controller must not provide a control action u_i that can be executed by another controller if it inhibits its ability to provide action u_j, for which it is critically needed. [UCCA 44.8]

SC-2.DA.4.1: N/A. Top-Level Scenario 1 does not apply to the UCCAs.

SC-2.DA.4.2: The team leadership must not direct a teammate to provide u_i, which can be executed by another controller, if that prevents the teammate from providing u_j, for which it is critically needed. [S-44.8.2] [DA, SA]

SC-2.DA.4.2.1: The team leadership must not direct a team of controllers as a whole to provide u_i if (1) it anticipates that the controller they will allocate is critically needed to provide u_j and (2) another controller can provide u_i. [S-44.8.2] [DA, SA]

SC-2.DA.4.3: A team of controllers must not allocate a specific controller to provide u_i if that controller is instead critically needed to provide u_j, and if another controller can provide u_i. [S-44.8.3] [DA, SA]

SC-2.DA.4.4: A controller that is critically needed to provide control action u_j must not divert its control effort to provide u_i without direction from team leadership. [S-44.8.4] [DA, SA]

SC-2.DA.4.4.1: The controller must be informed if it is critically needed to provide u_j a so that its efforts are not diverted to provide other control actions.

SC-2.DA.4.5: The team leadership must not provide u_i, if u_i can be delegated to a teammate, and if the team leadership is critically needed to provide u_j. [S-44.8.5] [DA, SA]

Safety Constraints Primarily Traced to Transfer of Authority (TA)

The following safety constraints are derived from Abstraction 2b Type 3-4 UCCAs and their top-level scenarios. They employ the same notation as the Level 2 constraints above.

SC-2.TA.1: A controller must not start providing a control action before another controller ends providing the same action if that creates an unsafe conflict. [UCCA 47.1] [TA, SA]

SC-2.TA.1.6: The team leadership must not direct one controller to start a control action before directing another controller to end the same action when that creates an unsafe conflict. [S-47.1.6] [TA, SA]

SC-2.TA.1.7: If controllers are properly directed by the team leadership to hand off a control action, then those controllers must not start early or end late their respective actions in a way that creates an unsafe conflict. [S-47.1.6] [TA, SA]

SC-2.TA.1.8: The team leadership must not direct a teammate to start or end a control action at a time that creates a conflict with how the team leadership hands off control to/from that controller.

***Note**: This requirement is not traced to a scenario in the analyzed system, but it is included in case a different version of the system architecture makes it applicable. For example, if the TL could provide a fix, then scenarios would be uncovered for which this requirement is necessary.*

SC-2.TA.1.8.1: The team leadership must not start its control action early compared to when it directs a teammate to end the same action if that leads to an unsafe conflict.

SC-2.TA.1.8.2: The team leadership must not end its control action late compared to when it directs a teammate to start the same action if that leads to an unsafe conflict.

SC-2.TA.2: A controller must not end providing a control action before another controller starts providing the same action if that creates an unsafe gap in execution. [UCCA 48.2, UCCA 56.3] [TA, SA]

SC-2.TA.2.1: The team leadership must not direct one teammate to end a control action before the time it directs another teammate to start the same action when that creates an unsafe gap in execution. [S-48.2.6] [TA, SA]

SC-2.TA.2.2: If a set of controllers is directed by the team leadership to hand off control in a safe way, then those controllers must not end early or start late their respective actions when that creates an unsafe gap in execution. [S-48.2.7, S-56.3.7] [TA, SA]

SC-2.TA.2.3: The team leadership must not direct a teammate to provide a control action in a way that creates an unsafe gap in control with how the team leadership hands off control to/from that controller. [S-56.3.8] [TA, SA]

SC-2.TA.2.3.1: The team leadership must not start its control action late relative to the time it directs a teammate to end the same action if that leads to an unsafe gap. [S-56.3.8] [TA, SA]

SC-2.TA.2.3.2: Team leadership must not end its control action early relative to the time it directs a teammate to start the same action if that leads to an unsafe gap. [S-56.3.8] [TA, SA]

Safety Constraints Primarily Traced to Shared Authority (SA)

The following safety constraints are derived from Abstraction 2a UCCAs and their top-level scenarios. They employ the same notation as the Level 2 constraints above.

SC-2.SA.1: Controllers on the team must not provide lower-priority control actions that prevent them from providing higher-priority actions. [UCCA 10.5, UCCA 11.6] [SA]

SC-2.SA.1.1: The team leadership must direct controllers on the team to provide higher priority control actions if lower priority actions are also specified. [S-10.5.1, S-11.6.1] [SA]

SC-2.SA.1.2: The team leadership must not direct controllers on the team to provide control actions that prevent the team from providing other higher-priority actions. [S-10.5.2, S-11.6.2] [SA]

SC-2.SA.1.2.1: The team leadership must not direct a teammate to provide a control action if (1) that prevents it from providing one of the multiple coupled actions that are together of higher priority and (2) no other controller is capable or available to provide that action.

SC-2.SA.1.3: If controllers on the team are properly directed by the team leadership to provide high-priority control actions, the controllers must prioritize providing those actions over lower-priority ones, regardless of whether or not they were also directed to provide the lower-priority actions [S-10.5.3, S-10.5.4, S-11.6.3, S-11.6.4] [SA]

SC-2.SA.1.3.1: If a controller is assigned to provide a control action that prevents it from providing a higher priority action for which it is needed, that controller must inform the allocation authority so that it can be retasked. [S-10.5.3, S-11.6.3] [SA]

SC-2.SA.1.3.2: If a controller is assigned to provide a control action that prevents it from providing a lower priority action for which it was also assigned, that controller must inform the allocation authority so the lower priority action can be reassigned. [S-10.5.3, S-11.6.3] [SA]

SC-2.SA.1.4: see SC-2.SA.1.3

SC-2.SA.1.5: The team leadership must provide the higher priority control actions if it is needed for those actions. [S-10.5.5] [SA]

SC-2.SA.1.5.1: The team leadership must not provide lower priority control actions if it prevents it from providing the higher priority ones. [S-10.5.5] [SA]

SC-2.SA.1.5.2: The team leadership must direct controllers to provide the necessary control actions coupled with their own. [S-11.6.5] [SA]

SC-2.SA.2: Controllers that are dependent on one another to provide coupled control actions must provide their respective control actions. [UCCA 3.3, UCCA 2.2] [SA]

SC-2.SA.2.1: The team leadership must direct all controllers involved in coupled and mutually dependent control actions to provide their respective actions. [S-3.3.1, S-2.2.1] [SA]

SC-2.SA.2.2: Controllers must inform the team leadership if the directions they were provided for coupled control actions are not properly specified. Specifically:

SC-2.SA.2.2: Controllers must inform the team leadership if a subset of the coupled control actions is not properly specified or directed. [S-3.3.2, S-2.2.2] [SA]

SC-2.SA.2.3: If multiple controllers on the team are properly directed to provide coupled control actions, those controllers must provide those actions together. [S-3.3.3, S-2.2.3] [SA]

SC-2.SA.2.3.1: If one of the controllers is unable to provide its control action, that controller must inform the other controllers involved and the team leadership. [S-3.3.3, S-2.2.3] [SA]

SC-2.SA.2.3.2: If one of the controllers is unable to provide its control action, the other controllers that are dependent on it must not provide their control actions. [S-3.3.3, S-2.2.3] [SA]

SC-2.SA.2.4: If multiple controllers on the team are properly directed to provide coupled control actions, the controllers on the team that are not directed must not provide them. [S-3.3.4, S-2.2.4] [SA]

SC-2.SA.2.5: If the team leadership is responsible for providing control actions that are coupled to the actions directed to other controllers, it must provide those control actions consistently with those directions. [S-3.3.5, S-2.2.5] [SA]

SC-2.SA.2.5.1: If the team leadership is unable to provide those actions, it must modify its directions to the controllers that depend on it for coupled execution. [S-3.3.5, S-2.2.5] [SA]

SC-2.SA.3: Controllers that are responsible for jointly controlling a subprocess must provide control actions to the same subprocess. [UCCA 4.4] [SA]

SC-2.SA.3.1: N/A. Top-Level Scenario 1 does not apply to the UCCA.

SC-2.SA.3.2: The team leadership must specify the same subprocess to every controller it directs to jointly control, including itself. [S-4.4.2] [SA]

SC-2.SA.3.3: If multiple controllers are properly directed to jointly control a subprocess, those controllers must provide those actions to that process. [S-4.4.3] [SA]

SC-2.SA.3.3.1: If controllers are directed to jointly control an unspecified or inconsistently specified subprocess, those controllers must notify the team leadership. [S-4.4.3] [SA]

SC-2.SA.3.3.2: If controllers are directed to jointly control an unspecified or inconsistently specified subprocess, those controllers must not provide their control actions. [S-4.4.3] [SA]

SC-2.SA.3.4: N/A. Top-Level Scenario 4 does not apply to the UCCA.

SC-2.SA.3.5: If the team leadership is responsible for jointly controlling a subprocess, it must provide its control actions to the same subprocess that it directed the other controllers to act on. [S-4.4.5] [SA]

SC-2.SA.4: Controllers that depend on one another to provide coupled control actions must start and end their actions in an adequate sequence. [UCCA 15.1, UCCA 17.2] [SA]

SC-2.SA.4.6: The team leadership must direct the controllers in a way that does not conflict with the adequate sequence of starting and ending their actions. [S-15.1.6, S-17.2.6] [SA]

SC-2.SA.4.7: If multiple controllers on the team are properly directed to provide coupled control actions, those controllers must start and end their control actions in the adequate relative sequence. [S-15.1.7, S-17.2.7] [SA]

SC-2.SA.4.8: The team leadership must start and end its control actions in an adequate sequence relative to how it specified the start and end of coupled control actions it directed other controllers to provide. [S-15.1.8, S-17.2.8] [SA]

Safety Constraints Primarily Traced to Mutually Closing Control Loops (MC)

The following safety constraints are derived from the refined scenarios that relate to the *mutually closing control loops* collaborative control dynamic. The labeling follows a similar convention as the one described for the *dynamic authority* safety constraints above. The only difference is that

the digits do not reference a top-level scenario. The traceability of these safety constraints to DA, TA, and/or SA is implied and determined by the scenarios to which they are linked. For example, safety constraints that are traced to scenario S-37.1.1 are, therefore, also traceable to DA and SA.

SC-2.MC.1: *Mutually Closing Control Loops – Feedback about Controlled Process from Collaborators:* A controller that relies on a teammate to receive feedback on its control action(s) to the shared process must be able to adequately receive and accurately interpret that feedback. Specifically:

SC-2.MC.1.1: The controller must be able to interpret which control action the feedback from the teammate pertains to. [MC]

SC-2.MC.1.1.1: The controller must be able to interpret which controlled subprocess the feedback from the teammate pertains to. [S-37.1.3, S-40.4.3, S-40.4.5, S-1.1.3, S-1.1.5, S-4.4.3, S-4.4.5] [MC]

SC-2.MC.1.1.2: The controller must be able to interpret which controller the feedback from the teammate pertains to. [S-37.1.3, S-40.4.3, S-40.4.5, S-1.1.3, S-1.1.5, S-17.2.7] [MC]

SC-2.MC.1.2: Feedback exchanged between two different types of controllers (e.g., a human controller and a machine controller) must adhere to a syntax that enables semantic alignment between the two. [S-37.1.3, S-38.2.3, S-40.4.3, S-40.4.5, S-1.1.3, S-1.1.5, S-3.3.3] [MC]

SC-2.MC.1.3: The human interface must allow a human controller under heavy workload to accurately specify the feedback provided to a machine controller. [S-37.1.3, S-38.2.3, S-1.1.3] [MC]

SC-2.MC.1.4: The human interface must allow a human controller under heavy workload to accurately and timely receive feedback provided by a machine controller. [S-40.4.5, S-1.1.5] [MC]

SC-2.MC.1.5: A controller must not misinterpret feedback provided by a teammate as a need to provide (or not provide) a control action that is not necessary (or is necessary). [S-37.1.3, S-38.2.3, S-43.7.3, S-1.1.3, S-10.5.3, S-3.3.3, S-3.3.4, S-3.3.5, S-2.2.3, S-2.2.4, S-11.6.3] [MC]

SC-2.MC.1.6: A controller must be able to determine which teammate provided the feedback. [S-37.1.3, S-38.2.3, S-1.1.3] [MC]

SC-2.MC.1.7: A controller must not act on unsolicited feedback from a teammate by providing (or not providing) a control action without first determining that action is safe and follows the intent of team leadership. [S-38.2.3, S-39.3.3, S-41.5.3, S-41.5.4, S-41.5.5, S-42.6.3, S-42.6.4, S-42.6.5] [MC]

SC-2.MC.1.8: The timing and intensity with which a machine controller provides feedback must not lead the receiving human controller to misinterpret the meaning of the feedback. [S-40.4.3, S-40.4.5, S-1.1.5] [MC]

SC-2.MC.1.9: The controller must be configured and capable to receive feedback from the teammate. [S-40.4.3, S-40.4.5, S-1.1.5, S-15.1.7, S-15.1.8, S-17.2.7] [MC]

SC-2.MC.1.9.1: If the controller is not configured or capable to receive feedback from the teammate, then it must inform the controller responsible for task allocation. [S-40.4.3, S-40.4.5, S-1.1.5] [MC]

SC-2.MC.1.10: A controller must not change its assessment of task prioritization based only on feedback provided by a teammate without first determining that the change is aligned with the intent of team leadership. [S-44.8.3, S-10.5.3, S-11.6.5] [MC]

SC-2.MC.1.11: A controller must be able to determine if a channel of feedback provided by a teammate is degraded to recognize when data may have been lost in the transfer. [S-44.8.5] [MC]

SC-2.MC.1.12: A controller must not misinterpret feedback provided by a teammate as a need to change how or when a control action handoff occurs with another controller. [S-47.1.7, S-48.2.7] [MC, TA]

SC-2.MC1.13: A controller must not act on feedback from a teammate without first determining if a change in its planned execution impacts the execution of other controllers (e.g., in coupled tasks). [S-10.5.3, S-10.5.4, S-10.5.5] [MC, SA]

SC-2.MC.2: *Mutually Closing Control Loops – Feedback about Collaborator Control Actions from Controlled Process:* A controller that relies on feedback from the controlled process generated in response to a teammate's control action must be able to adequately and accurately interpret that feedback. Specifically:

SC-2.MC.2.1: The controller must verify that the feedback is in response to the expected control action provided by a teammate. [MC]

SC-2.MC.2.1.1: The controller must verify that the feedback is in response to the expected controlled subprocess. [S-40.4.3, S-40.4.5, S-1.1.5, S-4.4.3, S-4.4.5, S-15.1.7, S-15.1.8] [MC]

SC-2.MC.2.1.2: The controller must verify that the feedback is in response to the expected teammate. [S-37.1.3, 2.2.3, S-1.1.3, S-3.3.3, S-3.3.4, S-3.3.5, S-15.1.7, S-15.1.8] [MC]

SC-2.MC.2.2: The controller and the teammate must be coordinated on when and how the teammate will provide its control action so that the controller can adequately receive the feedback. [S-40.4.3, S-40.4.5] [MC]

SC-2.MC.2.3: The controller must be configured and capable to sense the feedback signal generated in response to a control action provided by the teammate to the process. [S-40.4.3, S-40.4.5, S-1.1.5] [MC]

SC-2.MC.2.3.1: If the controller is not configured or capable to sense the feedback signal, then it must inform the controller responsible for task allocation. [S-40.4.3, S-40.4.5, S-1.1.5] [MC, DA]

SC-2.MC.2.4: A controller must not act on any unexpected feedback from the process that is generated in response to another controller's actions without first determining that the action is safe and is aligned with the intent of team leadership. [S-41.5.3, S-41.5.4, S-41.5.5, S-42.6.3, S-42.6.4, S-42.6.5] [MC]

SC-2.MC.3: A controller must be able to interpret how feedback provided by a teammate or the process relates to an expected control handoff with another controller. [S-47.1.7, S-48.2.7] [MC]

SC-2.MC.3.1: A controller must not claim/reclaim control in a way that conflicts with a planned handoff based only on third-party feedback (e.g., a controller that was not intended for the handoff or the process). [S-47.1.7] [MC]

SC-2.MC.3.2: A controller must not relinquish control (early) in a way that conflicts with a planned handoff based only on third-party feedback (e.g., a controller that was not intended for the handoff or the process). [S-48.2.7] [MC]

SC-2.MC.4: *Mutually Closing Control Loops – Feedback about Controlled Process from Collaborators:* A controller must not change how it hands off control with a teammate only on the basis that it provided feedback to the teammate. Closed-loop coordination between the two controllers must first occur to ensure the feedback is received and that it is safe to make the change. [S-56.3.8] [MC]

SC-2.MC.5: *Mutually Closing Control Loops – Feedback about Controlled Process from Collaborators:* The feedback provided by a controller regarding a teammate's control action to the process must specify the subprocess from which the feedback was received. [S-4.4.3] [MC]

Safety Constraints Primarily Traced to Cognitive Alignment (CA) and Lateral Coordination (LC)

The following safety constraints are derived from the refined scenarios that relate to the *cognitive alignment* and *lateral coordination* collaborative control dynamics. The labeling follows the same convention specified for *mutually closing control loops* above.

SC-2.CA.1: ***Construction***: A controller on the team must have a control algorithm that is developed and configured consistently with the other controllers it is working with. Specifically:

SC-2.CA.1.1: Its control algorithm configuration must be consistent with the control algorithm of the team leadership.

SC-2.CA.1.1.1: The control algorithm must be consistent with how the team leadership directs the team to provide control actions. [S-37.1.3, S-40.4.3, S-40.4.5, S-43.7.3, S-44.8.4, S-1.1.5, S-10.5.5, S-3.3.3, S-2.2.3, S-2.2.5] [CA]

SC-2.CA.1.1.2: The control algorithm must be consistent with how the team leadership expects feedback for its directions. [S-41.5.5, S-42.6.5, S-44.8.5, S-10.5.4, S-10.5.5] [CA]

SC-2.CA.1.1.3: The control algorithm must be consistent with how the team leadership prioritizes the control actions it directs the team to provide. [S-44.8.3, S-1.1.3, S-10.5.3, S-10.5.3, S-11.6.3] [CA]

SC-2.CA.1.1.4: The control algorithm must be consistent with how the team leadership defines execution timelines and handoffs between controllers. [S-47.1.7, S-48.2.7, S-56.3.7, S-56.3.8, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.1.2: Its control algorithm configuration must be consistent with the control algorithm of peer controllers with which it shares the controlled process.

SC-2.CA.1.2.1: The control algorithm must be consistent with those of controllers that provide coupled control actions. [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3, S-3.3.3, S-2.2.3, S-11.6.3, S-4.4.5] [CA]

SC-2.CA.1.2.2: The control algorithm must be consistent with those of controllers that can provide the same control action to prevent assignment conflicts or facilitate handoffs. [S-38.2.3, S-39.3.3, S-41.5.3, S-41.5.4, S-42.6.3, S-42.6.4, S-43.7.3, S-47.1.7, S-48.2.7, S-56.3.7, S-56.3.8] [DA, CA]

SC-2.CA.1.2.3: The control algorithm must be consistent between controllers regarding the method used to assign the different types of control actions (e.g., fully specified by team leadership, distributed task allocation, self-authorized). [S-44.8.3, S-10.5.3, S-3.3.4, S-2.2.4, S-11.6.3] [CA]

SC-2.CA.2: ***Initialization***: Controllers involved in distributed decision-making over the control of a shared process must be provided with planning information that is consistent with each other. Specifically:

SC-2.CA.2.1: All Controllers involved in coupled control actions must be provided with consistent planning information for those actions. Specifically: [CA]

SC-2.CA.2.1.1: Control action requirements must be specified consistently. [S-37.1.3, S-40.4.3, S-43.7.3, S-47.1.7, S-48.2.7, S-56.3.7, S-56.3.8, S-1.1.3, S-10.5.3, S-3.3.3, S-2.2.3, S-11.6.3] [CA]

SC-2.CA.2.1.2: The responsibilities and status of the controllers involved in the control actions must be consistent. [S-37.1.3, S-38.2.3, S-39.3.3, S-40.4.3, S-41.5.3, S-42.6.3, S-43.7.3, S-44.8.5, S-47.1.7, S-48.2.7, S-56.3.7, S-56.3.8, S-1.1.3, S-10.5.3, S-3.3.4, S-2.2.4, S-11.6.3] [CA, DM]

SC-2.CA.2.1.3: The subprocesses the control actions are specified for must be consistent. [S-4.4.3] [CA]

SC-2.CA.2.1.4: The way execution timelines are initialized must be consistent. [S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.2.2: Controllers involved in coupled control actions must be able to detect and resolve differences in information regarding those actions. This includes the type of information listed above (see SC-2.CA.2.1). [S-37.1.3, S-40.4.3, S-3.3.3, S-2.2.3] [CA]

SC-2.CA.2.3: An automated controller must provide transparency to help human controllers understand how the automation allocates control actions to controllers. [S-40.4.5, S-41.5.5, S-42.6.5, S-1.1.5, S-2.2.5] [CA]

SC-2.CA.2.3.1: An automated controller must inform the team leadership of pre-programmed control behaviors relative to the shared process. [S-10.5.4] [CA]

SC-2.CA.2.4: Control actions coupled between human and machine controllers must be defined and bounded by a protocol understood by the human, and to which the machine is developed to handle. [S-40.4.5, S-41.5.5, S-42.6.5, S-1.1.5, S-3.3.5, S-2.2.5, S-4.4.5] [CA]

SC-2.CA.2.5: All controllers involved in distributed task allocation must be provided with planning information that is consistent with one another. This includes the same items as listed under SC-2.CA.2.1. [S-43.7.3, S-1.1.3] [CA]

SC-2.CA.2.6: All controllers involved in coupled control actions or distributed task allocation must designate controllers and control actions by common and unique identifiers. [S-1.1.3] [CA]

SC-2.CA.2.7: All controllers involved in distributed task allocation must be consistently directed on how to prioritize control actions allocated to a controller directly by the team leadership versus those delegated by the team leadership to be allocated. [S-1.1.3, S-11.6.3] [CA]

SC-2.CA.3: ***Model Updates – Vertical Coordination*:** Updates provided by the team leadership to controllers on the team must not inhibit the team from properly executing coordinated actions. Specifically:

SC-2.CA.3.1: Updates provided by team leadership must be consistent between the controllers involved in coupled control actions. Specifically:

SC-2.CA.3.1.1: The team leadership must inform all controllers involved in the coupled control if updates are made to the specification of any of those actions. [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3, S-3.3.3, S-2.2.3, S-4.4.5] [CA]

SC-2.CA.3.1.2: The team leadership must inform all controllers involved in the coupled control if one of the actions is reassigned to a different controller. [S-1.1.3, S-10.5.3] [CA]

SC-2.CA.3.1.3: The team leadership must specify consistently how the controllers need to coordinate their coupled actions. [S-3.3.3, S-2.2.3] [CA]

SC-2.CA.3.1.4: The team leadership must specify consistently which subprocesses the assigned control actions apply to. [S-4.4.3, S-4.4.5] [CA]

SC-2.CA.3.2: The team leadership must not provide information that is inconsistent with how controllers on the team refine their coordinated plan unless it intends to override their refined plan. The refined coordinated plans include:

SC-2.CA.3.2.1: Coordination of coupled control actions [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3, S-3.3.3, S-2.2.3, S-4.4.5] [CA]

SC-2.CA.3.2.2: Coordination of task handoffs [S-47.1.7, S-48.2.7] [CA]

SC-2.CA.3.3: The team leadership must not provide unnecessary mission planning updates.

SC-2.CA.3.3.1: The team leadership must not unnecessarily update the roles and responsibilities of the controllers such that it interferes with them accomplishing otherwise safe plans. [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3, S-3.3.3, S-2.2.3] [CA]

SC-2.CA.3.3.2: Human team leaders must be trained to not excessively manage automated controllers at the expense of fulfilling their own control responsibilities. [S-40.4.5, S-1.1.5, S-2.2.5] [CA]

SC-2.CA.3.3.3: The team leadership must not unnecessarily update the specifications of control actions (e.g., execution timelines, handoff times) such that it interferes with the otherwise safe execution of controllers. [S-56.3.8, S-1.1.3, S-10.5.3] [CA]

SC-2.CA.3.4: The team leadership must be able to specify low-level execution parameters for a control action without changing the controller allocated for that action. [S-41.5.5, S-42.6.5]

SC-2.CA.3.5: The team leadership must be informed why a distributed task allocation process does not assign a controller to a control action as intended. Specifically:

SC-2.CA.3.5.1: The team leadership must be informed if the actions as specified impose constraints that cannot be satisfied by the team. [S-43.7.3, S-1.1.3, S-10.5.3, S-11.6.3] [CA]

SC-2.CA.3.5.2: The team leadership must be informed if specific controllers have existing allocation constraints that prohibit them from being assigned to provide a control action. [S-43.7.5, S-56.3.7, S-1.1.5] [CA]

SC-2.CA.3.6: The team leadership must have the ability to assign a specific controller to specific control action, even if a distributed allocation process is available. Specifically:

SC-2.CA.3.6.1: The team leadership must have the ability to override the distributed allocation and assign specific controllers to specific actions. [S-44.8.5] [CA]

SC-2.CA.3.6.2: The intent of the team leadership to assign a specific controller to an action or to delegate the allocation of that action to the distributed process must be explicit. [S-44.8.3] [CA]

SC-2.CA.4: ***Model Updates – Lateral Coordination (Communication)*****:** Controllers must be able to actively communicate as necessary to coordinate the execution of their control actions.

SC-2.CA.4.1: Communication channels between controllers must have sufficient capacity to support coordination messages at the time of execution while overcoming [TBD] environmental degradations (e.g., fading, jamming). [S-37.1.3, S-40.4.3, S-43.7.3, S-47.1.7, S-48.2.7, S-56.3.8, S-1.1.3, S-10.5.3, S-3.3.3, S-2.2.3, S-11.6.3, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA, LC]

SC-2.CA.4.1.1: Communication channels between controllers must ensure that as they degrade, they do not deliver messages with altered content. [S-4.4.3] [CA]

SC-2.CA.4.2: Semantics must be specified to facilitate human-machine communication over the coordination of coupled control actions. [S-37.1.3, S-40.4.3, S-40.4.5, S-56.3.8, S-1.1.3, S-1.1.5, S-10.5.3, S-3.3.3, S-2.2.3, S-2.2.5, S-4.4.5] [CA, LC]

SC-2.CA.4.3: Machine controllers must not interrupt the ability of a human controller to provide her/his control actions by requesting information with [TBD unreasonable] frequency. [S-37.1.3, S-40.4.3, S-40.4.5, S-56.3.8, S-1.1.3, S-1.1.5, S-10.5.3, S-2.2.5] [CA, LC]

SC-2.CA.4.4: The time at which information is produced must be appended to coordination messages between controllers and accounted for in decision-making. The type of information this applies to includes:

SC-2.CA.4.4.1: State estimates [S-37.1.3, S-40.4.3, S-43.7.3, S-1.1.3, S-10.5.3, S-11.6.3, S-4.4.5] [CA, LC]

SC-2.CA.4.4.2: Model variables describing the controlled process [S-40.4.5, S-1.1.5, S-10.5.4, S-2.2.5] [CA, LC]

SC-2.CA.4.4.3: Control action time-windows and handoff times [S-47.1.7, S-48.2.7, S-56.3.7] [CA, LC]

SC-2.CA.4.5: A controller must identify what subprocess it refers to in coordination messages provided to other controllers (related to SC-2.MC.1.1.1**)**. [S-37.1.3, S-40.4.3, S-40.4.5, S-44.8.3, S-1.1.3, S-1.1.5, S-10.5.3, S-10.5.5, S-3.3.3, S-2.2.3, S-2.2.5, S-4.4.5] [CA, LC]

SC-2.CA.4.5.1: The controller must identify subprocesses by their common unique identifiers. [S-1.1.3] [CA, LC]

SC-2.CA.4.5.2: The controllers must inform other controllers if it has [TBD level] uncertainty in correlating a subprocess to that specified by other controllers. [S-4.4.3] [CA]

SC-2.CA.4.6: A controller must not change its control assignment for control action u_i based on coordination messages alone. The controller must first determine if the change is aligned

with the intent of the team leadership [S-38.2.3, S-39.3.3, S-41.5.3, S-41.5.4, S-41.5.5, S-42.6.3, S-42.6.4, S-42.6.5, S-47.1.7, S-48.2.7, S-3.3.4, S-2.2.4] [CA, LC]

SC-2.CA.4.7: Controllers must be able to identify and report to the team leadership when and how the information shared between controllers is too inconsistent to coordinate execution. [S-44.8.5] [CA, LC]

SC-2.CA.4.8: A controller must not share coordination information that conflicts with the information specified by the team leadership. Specifically:

SC-2.CA.4.8.1: A controller must not communicate any internal manipulations of control specifications that it makes for its own internal decisions (e.g., local prioritization, local time-window determination, etc…). [S-11.6.3] [CA]

SC-2.CA.4.9: Controllers must use a common, universal time reference with a standardized format in coordination. [S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.5: ***Model Updates – Lateral Coordination (Observation)*:** A controller must not act on a prediction of another controller's intent using only observations of that controller's behavior.

SC-2.CA.5.1: A controller must confirm the intent of a controller using means other than observation only (e.g., communications, vertical coordination) before taking action accordingly. Specifically:

SC-2.CA.5.1.1: A controller must not act by changing its planned execution or control actions. [S-37.1.3, S-38.2.3, S-39.3.3, S-40.4.3, S-41.5.3, S-42.6.3, S-43.7.3, S-44.8.3, S-47.1.7, S-48.2.7, S-56.3.8, S-1.1.3, S-10.5.3, S-10.5.5, S-3.3.4, S-2.2.4, S-11.6.3, S-4.4.3, S-4.4.5] [CA, LC]

SC-2.CA.5.1.2: Team leadership must not act by changing the assignment of control actions to the controllers on the team. [S-41.5.4, S-42.6.4] [CA, LC]

SC-2.CA.5.1.3: A controller must not start a control action before confirming the intent or readiness of another controller to provide the coupled action. [S-3.3.3, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA, LC]

SC-2.CA.5.2: Any controller c_i must not expect any collaborator c_j to update its execution plan based on the observations c_j should make of c_i alone. c_i must explicitly declare its intent using other means (e.g., communications, vertical coordination). [S-40.4.5, S-1.1.5, S-2.2.5, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA, LC].

SC-2.CA.6: ***Model Updates – Prediction*:** A controller that makes control decisions based only on the predicted future states of other controllers must validate its predictions. This includes:

SC-2.CA.6.1: If a controller uses the current allocation of control responsibilities to predict the future states of its teammates, then those predictions must be revalidated, specifically when:

SC-2.CA.6.1.1: Predictions must be reassessed if changes occur in the allocation of control or the execution of those control actions. [S-43.7.3, S-56.3.7] [CA]

SC-2.CA.6.1.2: Predictions must be reassessed before a controller provides a control action coupled to the predicted state of another controller. [S-3.3.3, S-2.2.3, S-4.4.3] [CA]

SC-2.CA.6.1.3: A controller c_i must explicitly confirm the execution timeline of another controller c_j, which is responsible for a coupled control action, if timing information was not specified by the team leadership to c_i. [S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.6.2: The team leadership must not assume that a critical and taskable control action will be provided without being deliberately directed (e.g., will occur naturally as part of another control effort). [S-43.7.5, S-1.1.5, S-11.6.5] [CA]

SC-2.CA.6.3: The team leadership must review and understand control plans proposed by the automated controllers before approving them. [S-10.5.4] [CA]

SC-2.CA.6.4: The team leadership must be able to assign control actions based on the predicted future states of controllers.

SC-2.CA.6.4.1: The team leadership must be informed as soon as the system detects that the states of the controllers no longer satisfy the predictions. [S-10.5.5] [CA]

SC-2.CA.6.5: A controller must not provide a control action based only on a prediction that it would have been directed to act if it had connectivity to the team leadership. [S-3.3.4, S-2.2.4, S-11.6.5] [CA]

SC-2.CA.6.6: If able, a controller must continue to monitor the behavior of a teammate involved in a coupled control action, even if it has high confidence in that teammate. [S-4.4.5] [CA]

SC-2.CA.7: ***Decision-Making***: The decision-making process multiple controllers use collectively must provide adequate and actionable solutions for the team to execute. This includes:

SC-2.CA.7.1: The decision-making process must produce a feasible solution within a timeframe that allows the controllers to act. [S-37.1.3, S-40.4.3, S-40.4.5, S-48.2.7, S-56.3.8, S-1.1.5, S-10.5.3, S-2.2.5, S-11.6.3] [CA]

SC-2.CA.7.1.1: The decision-making process must update execution timelines for a dynamically controlled process in a timeframe that allows the controllers to act. [S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.7.1.2: If the decision-making process is unable to provide a solution within a timeframe that allows the controllers to act, it must inform the team leadership and the controllers involved in [TBD] time.

SC-2.CA.7.2: The decision-making process must not unnecessarily change its solution when that prohibits controllers from executing otherwise safe plans. This includes:

SC-2.CA.7.2.1: A solution to execute coupled control actions, including execution timelines. [S-37.1.3, S-40.4.3, S-10.5.3, S-3.3.3, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.7.2.2: A solution to allocate control actions [S-43.7.3, S-10.5.3] [CA]

SC-2.CA.7.2.3: A solution to execute control handoffs [S-47.1.7, S-56.3.7] [CA]

SC-2.CA.7.3: The decision-making process must be able to detect and take appropriate measures to handle unexpected and outlier decision outputs by some of the controllers. [S-37.1.3, S-40.4.3, S-43.7.3, S-47.1.7, S-56.3.7, S-10.5.3] [CA]

SC-2.CA.7.4: The decision-making process must be able to identify and handle factors that cause controllers to deviate from the decided execution solution. Specifically:

SC-2.CA.7.4.1: A controller must inform the system if it lacks confidence in the ability of other controllers to carry out their responsibilities. [S-37.1.3, S-40.4.3, S-40.4.5, S-1.1.5, S-10.5.3, S-3.3.3, S-2.2.3, S-2.2.5] [CA]

SC-2.CA.7.4.2: A controller must inform the system if it has contextual factors that prevent it from carrying out its responsibilities or changing its execution timelines. [S-37.1.3, S-40.4.3, S-43.7.3, S-10.5.3, S-3.3.3, S-2.2.3, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.7.5: The decision-making process must incorporate an arbitration mechanism to resolve conflicts or lack of consensus, produce a safe and actionable plan, or inform the team leadership no plan is possible. [S-1.1.3] [CA]

SC-2.CA.7.5.1 The arbitration mechanism must assign a controller that is capable of executing the control action. [S-38.2.3] [CA]

SC-2.CA.7.5.2 The arbitration mechanism must ensure that only one controller is assigned to a control action. [S-39.3.3, S-42.6.3] [CA]

SC-2.CA.7.5.3 The arbitration mechanism must account for the prioritization of the control actions. [S-44.8.3] [CA]

SC-2.CA.7.5.4 The arbitration mechanism must specify the ordered sequence of the control actions. [S-1.1.3] [CA]

SC-2.CA.7.5.5 The arbitration mechanism must ensure that coupled control actions are allocated for coupled execution. [S-4.4.3] [CA]

SC-2.CA.7.6: If the decision-making process is unable to produce a solution:

SC-2.CA.7.6.1 The decision-making process must explain to the team leadership why it is unable to produce a solution (e.g., computational complexity, no feasible solution, timed out). [S-43.7.3, S-10.5.3] [CA]

SC-2.CA.7.6.2 The decision-making process must recommend courses of action (e.g., simplify the problem, propose a non-optimal solution, and allow more time to run). [S-43.7.3, S-10.5.3] [CA]

SC-2.CA.7.7: The decision-making process must account for factors of interest in determining execution solutions. Specifically:

SC-2.CA.7.7.1: It must account for the priority of control actions when producing control allocation decisions. [S-44.8.3] [DA, CA]

SC-2.CA.7.7.1.1: It must account for the priority of coupled control actions as a whole. [S-11.6.3] [DA, CA]

SC-2.CA.7.7.2: It must account for environmental disturbances that may affect execution timelines involved in control handoffs. [S-48.2.7, S-56.3.8] [CA]

SC-2.CA.7.7.3: It must ensure consensus is reached on designated key parameters (e.g., the specific subprocess controlled in coupled tasks), and not allow consensus only on secondary parameters to be sufficient to reach a solution. [S-4.4.5] [CA]

SC-2.CA.7.8: The decision-making process must be able to provide the team leadership with a reasonable estimate of how long it will take to produce a solution. [S-1.1.3] [CA]

SC-2.CA.7.9: The decision-making process must track previously explored infeasible solutions to make sure they are not repeated and the algorithm converges. This includes:

SC-2.CA.7.9.1: The process must eliminate control allocations found to be infeasible from further consideration. [S-1.1.3] [CA]

SC-2.CA.7.9.2: The process must inform the team leadership if control allocations are found to be infeasible. [S-1.1.3] [CA]

SC-2.CA.7.9.3: No controller may act on a control allocation found to be infeasible unless it is explicitly directed to do so by the team leadership. [S-10.5.4] [CA]

SC-2.CA.7.10: The decision-making process must not bias allocating certain types of control actions over others only because the algorithm is more efficient for such actions. [S-10.5.3] [CA]

SC-2.CA.7.11: The system must not auto-correct errors made by the team leadership without determining that the correction aligns with the intent of the team leadership. [S-10.5.4] [CA]

SC-2.CA.7.12: The decision-making process must incorporate a deadline by which all controllers involved in the decision must have reached consensus and acknowledged the solution, otherwise the solution is voided. [S-3.3.3, S-2.2.3] [CA]

SC-2.CA.7.13: The decision-making process must be applied only to the sub-processes that were specified by the team leadership. [S-3.3.4, S-2.2.4] [CA]

SC-2.CA.7.14: Each controller, when planning a decision, must employ the parameters directed by the team leadership or those shared between controllers. They may not substitute their own parameter values for planning unless those are coordinated with the other controllers. [S-4.4.3] [CA]

SC-2.CA.8: *Capacity*: The system must be able to identify and isolate controllers that are unable to coordinate with the rest of the team and hinder the team's ability to make decisions or issue collective control actions. Specifically:

SC-2.CA.8.1: The system must be able to identify controllers that are unable to meet coordination messaging requirements with other controllers.

SC-2.CA.8.1.1: The system must isolate controllers that disrupt coordination between the other controllers. [S-37.1.3, S-40.4.3, S-43.7.3, S-44.8.3, S-47.1.7, S-48.2.7, S-56.3.7, S-1.1.3, S-10.5.3] [CA]

SC-2.CA.8.1.2: Controllers that have not met the coordination requirements for control actions (e.g., correlation of a subprocess for coupled execution, execution timelines) must not provide those actions. [S-4.4.5, S-15.1.7, S-15.1.8, S-17.2.7, S-17.2.8] [CA]

SC-2.CA.8.2: The system must be able to identify and isolate controllers that are unable to receive and process information to use in distributed decision-making. [S-38.2.3, S-39.3.3, S-42.6.3, S-10.5.4] [CA]

SC-2.CA.8.3: Automated controllers on the team must have a neutral backup behavior to execute if human controller(s) are unable to participate in the coordination process. [S-40.4.5, S-44.8.5, S-56.3.8, S-1.1.5, S-10.5.5, S-2.2.5] [CA]

SC-2.CA.8.4: The system must prevent human controllers from fixating on the execution of automated teammates at the expense of their performing their own actions. [S-43.7.5, S-1.1.5] [CA]

SC-2.CA.8.5: The system must be able to identify and inform the team leadership if the dimensionality of the planning problem exceeds the computational capacity of the system [S-1.1.3] [CA].

SC-2.CA.8.6: An automated controller must check the validity of its internal processes (i.e., system heartbeat check) before executing assigned control actions. [S-3.3.3, S-2.2.3] [CA]

SC-2.CA.8.7: The system must alert human operators if they attempt to carry out control actions for which a required coupled action is not allocated to a controller. [S-3.3.5] [CA]

SC-2.CA.8.8: The planning responsibilities delegated to controllers must remain within the computational bounds of those controllers. [S-11.6.3] [CA]

SC-2.CA.8.8.1: The system must ensure that it does not bias solving easier planning problems over those that are more complex, but still within the capacity of the controllers. [S-11.6.3] [CA]

SC-2.CA.9: ***Initialization:*** The process to specify control actions must be reasonably consistent for different types of actions to avoid biasing the team leadership to only assigning tasks that are easier to specify. [S-10.5.5] [CA]

Safety Constraints Primarily Traced to Dynamic Membership (DM)

The following safety constraints are derived from the refined scenarios that relate to the *dynamic membership* collaborative control dynamic. The labeling follows the same convention specified for *mutually closing control loops* above.

SC-2.DM.1: The system must notify the team leadership if a controller is added or removed from active participation status. Specifically:

SC-2.DM.1.1: The team leadership must be informed in advance [TBD time] when a controller is planned to be added or removed from the team. [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-44.8.2, S-1.1.1, S-3.3.5, S-4.4.2] [DM]

SC-2.DM.1.1.1: The team leadership must be informed if a controller is slated to execute a control action with a time window that goes beyond its planned time of removal from the team. [S-56.3.6] [DM]

SC-2.DM.1.2: The team leadership must be informed as soon as possible [TBD time] when a controller is added or removed from the team for unplanned reasons. [S-37.1.3, S-43.7.3, S-44.8.2, S-1.1.3, S-3.3.3, S-2.2.3, S-15.1.7, S-17.2.7, S-17.2.8] [DM]

SC-2.DM.1.3: The team leadership must be informed and take into consideration the capabilities of controllers added or removed from the team. [S-38.2.2, S-41.5.2] [DM]

SC-2.DM.1.4: The team leadership must be able to query other controllers about their intentions to add or remove a controller from the team. [S-40.4.5, S-1.1.5, S-10.5.5, S-2.2.5, S-11.6.5] [DM]

SC-2.DM.2: The team leadership must not deviate from nominal procedures directing the team in response to the dynamic membership of the team. Specifically:

SC-2.DM.2.1: The team leadership must not delay directing the team to provide a control action in anticipation of:

SC-2.DM.2.1.1: a new controller becoming available to provide it. [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-1.1.1, S-3.3.5, S-4.4.2] [DM]

SC-2.DM.2.1.2: the possibility that a controller may be dropping out. [S-1.1.1] [DM]

SC-2.DM.2.2: The team leadership must not direct the team to provide a control action in anticipation that a controller that is not yet part of the team will take it on. [S-44.8.2] [DM]

SC-2.DM.2.3: see SC-2.DA.1.2. [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-1.1.1, S-3.3.5, S-4.4.2] [DM]

SC-2.DM.2.4: If the team leadership must assign a specific controller to provide a control action, the team leadership must specify that controller even if it believes it is the only controller active on the team. [S-38.2.3, S-41.5.3, S-41.5.4, S-42.6.4, S-42.6.5] [DM]

SC-2.DM.3: The system must detect and take appropriate measures when a controller that was assigned a control action becomes inactive from the team. Specifically:

SC-2.DM.3.1: The system must inform the team leadership of the following:

SC-2.DM.3.1.1: how it is handling the control assignment, pending further direction from the team leadership (e.g., control assignment dropped, action remains assigned to disconnected controller). [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-1.1.1, S-1.1.2, S-10.5.2, S-10.5.3, S-3.3.1, S-3.3.5, S-2.2.1, S-11.6.3, S-4.4.2, S-4.4.3] [DM]

SC-2.DM.3.1.2: how the now inactive controller affects other controllers that were dependent on it for coupled control actions or control handoffs. [S-47.1.6, S-48.2.6, S-56.3.8] [DM]

SC-2.DM.3.1.3: highlight gaps or conflicts that arise with those dependent controllers (e.g., change control assignment, change handoff timeline). If possible, recommend courses of action to resolve those conflicts and gaps. [S-47.1.6, S-48.2.6] [DM]

SC-2.DM.3.2: The team leadership must be able to efficiently and properly update control assignments to overcome changes in participation from controllers on the team. [S-56.3.8, S-1.1.1, S-10.5.3, S-11.6.3, S-4.4.3] [DM]

SC-2.DM.3.2.1: The team leadership must update control assignment timelines (e.g., for coupled control execution, for control handoffs) as necessary to address changes in the participation of controllers on the team. [S-15.1.6, S-17.2.6] [DM]

SC-2.DM.3.3: The system must inform the other controllers assigned to provide coupled control actions or control handoffs about how the loss of the controller changes their

assignment. [S-37.1.3, S-47.1.7, S-48.2.7, S-1.1.3, S-3.3.3, S-2.2.3, S-15.1.7, S-17.2.7, S-17.2.8] [DM]

SC-2.DM.3.4: If the controller is reactivated, it must act in accordance with (1) updated direction from the team leadership regarding the control assignment, or (2) how the system informed the team leadership it would handle the control assignment if (1) does not occur. [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-1.1.1, S-10.5.3, S-3.3.5, S-11.6.3, S-4.4.2, S-4.4.3] [DM]

SC-2.DM.3.4.1: The controller must be informed of the updated direction from the team leadership as it rejoins the team before it issues any control actions that are assigned dynamically. [S-37.1.2, S-38.2.3, S-38.2.5, S-39.3.3, S-40.4.2, S-41.5.3, S-41.5.4, S-42.6.2, S-42.6.3, S-42.6.4, S-42.6.5, S-1.1.2, S-10.5.2, S-3.3.2, S-3.3.4, S-2.2.2, S-2.2.4] [DM]

SC-2.DM.3.4.2: When the controller rejoins under the authority of the team leadership, it must not provide any other control actions directed by other controllers without confirming that those align with the intent of the team leadership. [S-37.1.2, S-38.2.5, S-39.3.3, S-40.4.2, S-42.6.2, S-42.6.3, S-42.6.4, S-42.6.5, S-3.3.2, S-2.2.2] [DM]

SC-2.DM.3.5: If the controller was assigned using an automated process, the automation must attempt to reallocate the control action to another controller. [S-44.8.3, S-56.3.6] [DM]

SC-2.DM.3.5.1: The automation must ensure that the reallocation does not conflict (1) with higher priority assignment(s) of the other controller, or (2) with direct assignments by the team leadership (if consideration (2) is of importance). [S-44.8.3] [DM]

SC-2.DM.3.5.2: The automation must inform the team leadership if it is unable to reassign the control action to another controller. [S-56.3.6, S-56.3.8] [DM]

SC-2.DM.4: A controller outside the team must not direct a controller on the team without (1) receiving authorization from the team leadership, or (2) receiving authorization from higher authority and notifying the team leadership. Specifically:

SC-2.DM.4.1: It must not claim authority over a controller without (1) or (2). [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-1.1.1, S-3.3.5, S-4.4.2] [DM]

SC-2.DM.4.2: It must not remove a controller from the team that can execute its assigned tasks without (1) or (2) and informing the team leadership of the conflict the removal creates. [S-56.3.8, S-1.1.1, S-1.1.5, S-10.5.1] [DM]

SC-2.DM.5: The team leadership must be informed of the set of controllers active on the team.

SC-2.DM.5.1: The system must employ unique identifiers for each controller on the team and inform the team leadership of these identifiers (related to SC-2.CA.2.6**)**. [S-37.1.3, S-40.4.3, S-1.1.3, S-3.3.3, S-2.2.3, S-15.1.7, S-17.2.7, S-17.2.8] [DM]

SC-2.DM.5.2: The system must inform the team leadership if a controller is not operating in a mode that accepts the assignment of control actions. [S-37.1.3, S-40.4.3, S-1.1.3, S-3.3.3, S-2.2.3, S-15.1.7, S-17.2.7, S-17.2.8] [DM]

SC-2.DM.6: A controller on the team must not change its assigned control responsibilities based solely on the addition of a new controller. Specifically:

SC-2.DM.6.1: The controller must not drop its assigned control actions because a new controller is perceived to be better suited for the task. [S-37.1.3, S-40.4.3, S-1.1.3, S-3.3.3, S-2.2.3, S-15.1.7, S-17.2.7, S-17.2.8] [DM]

SC-2.DM.6.2: The controller must not change its planned control handoff because a new controller is perceived to be able to take over early. [S-56.3.7] [DM]

SC-2.DM.7: The team leadership must have control authority over the addition or removal of controllers from the team. Specifically:

SC-2.DM.7.1: The team leadership must be able to direct other controllers not to add or remove specific controller(s) from the team. [S-41.5.3, S-41.5.5, S-44.8.2, S-44.8.5, S-1.1.1, S-1.1.5, S-10.5.1] [DM]

SC-2.DM.7.2: The team leadership must be able to remove, or direct the system to remove, a specific controller from the team. [S-43.7.3, S-1.1.3] [DM]

SC-2.DM.8: The system must be able to handle the dynamic existence of subprocesses within the shared controlled process. Specifically:

SC-2.DM.8.1: The system must employ unique identifiers for each subprocess that are consistent across the controllers on the team (related to SC-2.CA.2.6**)**. [S-4.4.5] [DM]

SC-2.DM.8.2: The system must not automatically apply a control assignment specified for one (possibly stale) subprocess to another (possibly new) subprocess. [S-1.1.2] [DM]

SC-2.DM.8.3: The system must be able to account for subprocesses that are too volatile in presence to control, and instead, focus execution on those that are stable enough for the system to control. [S-1.1.3] [DM]

SC-2.DM.9: A controller must be added to the team in a state that does not require immediate direction from the team leadership. [S-1.1.5] [DM]

SC-2.DM.10: The team leadership must be able to specify a subset of the controllers over which the automated (or distributed) control allocation process assigns control actions. [S-11.6.1] [DM]

Safety Constraints Primarily Traced to Dynamic Connectivity (DC)

The following safety constraints are derived from the refined scenarios that relate to the *dynamic connectivity* collaborative control dynamic. The labeling follows the same convention specified for *mutually closing control loops* above.

SC-2.DC.1: The system must predict whether or not controllers will be disconnected from the team given the location the assigned control actions will be provided to their subprocesses.

SC-2.DC.1.1: The system must inform the team leadership if it predicts controller(s) to become disconnected as a result of the control assignment. [S-37.1.1, S-40.4.1, S-43.7.1, S-43.7.5, S-1.1.1, S-1.1.5, S-10.5.1, S-3.3.5, S-11.6.1, S-4.4.2] [DC]

SC-2.DC.1.2: The network connectivity prediction must account for the predicted location of controllers available to relay messages based on their assigned control responsibilities. [S-40.4.5, S-1.1.5, S-2.2.5] [DC]

SC-2.DC.2: The network must be able to relay communication messages (e.g., control and coordination) over multiple hops from the source to its destination. [S-37.1.1, S-40.4.1, S-43.7.1, S-1.1.1, S-10.5.1, S-3.3.1, S-3.3.5, S-11.6.1, S-4.4.2] [DC]

SC-2.DC.2.1: If the system is unable successfully relay a message to its destination, it must inform the source that the message was not transmitted. [S-37.1.1, S-40.4.1, S-43.7.1, S-1.1.1, S-10.5.1, S-3.3.1, S-3.3.5, S-11.6.1, S-4.4.2] [DC]

SC-2.DC.2.2: The system must allow the team leadership to cancel a control assignment, if needed, using other controller(s) as relays. [S-37.1.2, S-39.3.2, S-39.3.3, S-40.4.2, S-42.6.2, S-42.6.3, S-44.8.4, S-1.1.2, S-10.5.2, S-3.3.2] [DM]

SC-2.DC.2.3: The system must specify and authenticate the source, the destination, and the time of origin of information in all messages (relayed and direct). [S-37.1.3, S-40.4.3, S-43.7.5, S-56.3.7, S-1.1.5, S-10.5.3] [DM]

SC-2.DC.2.4: The controller specified as the destination of a message must treat a relayed message that originated from a specified source the same way as if the message was directly received from the source. [S-37.1.3, S-40.4.3, S-43.7.5, S-1.1.3, S-1.1.5, S-10.5.3] [DM]

SC-2.DC.2.5: A controller that is not specified as the destination of a message must not treat a message it receives as intended for itself. [S-38.2.3, S-41.5.3] [DM]

SC-2.DC.2.6: A controller specified as the destination that receives multiple different versions of control or coordination messages must consider the information with the most recent time of origin. [S-43.7.3, S-56.3.7, S-1.1.3, S-10.5.3] [DM]

SC-2.DC.2.7: The time at which a controller receives a message must not be used by itself as a timing reference for coordinated execution [S-15.1.7, S-17.2.7, S-17.2.8] [DC]

SC-2.DC.3: The system must have a protocol to handle the loss of communications between controllers. Specifically:

SC-2.DC.3.1: The loss of communication protocol must specify how long controllers can continue their nominal operating mode before reverting to an alternate behavior. [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3] [DC]

SC-2.DC.3.2: The loss of communication protocol must specify what the alternative behavior is if a controller enters loss of link from different types of nominal operating modes. [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3] [DC]

SC-2.DC.3.3: The system must inform the team leadership when controllers have entered a loss of link operating mode. [S-37.1.3, S-40.4.3, S-1.1.3] [DC]

SC-2.DC.3.4: The protocol must specify a process to transition controllers back from a loss of link mode to a nominal operating mode. [S-37.1.3, S-40.4.3, S-1.1.3, S-10.5.3] [DC]

SC-2.DC.3.5: A loss of link operating mode must not enable the controller to behave in a way that conflicts with the execution of the mission. Specifically:

SC-2.DC.3.5.1: The loss of link mode must not allow a controller to violate the intent of the team leadership (e.g., a controller must not provide a control action it is not authorized to provide by the team leadership). [S-41.5.4, S-42.6.4, S-3.3.4, S-2.2.4] [DC]

SC-2.DC.3.5.2: The loss of link mode must not lead a controller to issue control actions that exceed its control authority or capability. [S-41.5.4, S-42.6.4] [DC]

SC-2.DC.3.6: The system must assist human controllers execute the loss of link protocol (e.g., provide operating recommendations aligned with the alternative behavior). [S-56.3.6] [DC]

SC-2.DC.4: Multiple controllers that collaborate to provide coupled control actions or control handoffs must maintain a connectivity of [TBD] level with one another and of [TBD] level with the team leadership to carry out those responsibilities. [S-3.3.3, S-2.2.3] [DC]

SC-2.DC.5: Every controller must be informed which controllers it has connectivity with (whether direct or indirect connectivity) and those it does not. [S-11.6.5] [DC]

Safety Constraints Derived from Traditional STPA Feedback Control Factors

The following safety constraints are derived from the refined scenarios that relate to the traditional feedback control loop causal factors addressed in baseline STPA. Only a subset of the scenarios refined from these factors are included below for demonstration purposes. The labeling follows the same convention specified for *mutually closing control loops* above.

SC-2.X.1: ***Unsafe Control Input:*** Commands from higher authorities to the team leadership must be unambiguous and timely. Specifically:

SC-2.X.1.1: They must not be misinterpreted as a command not to direct the team to provide a control action. [S-37.1.1, S-40.4.1, S-40.4.5, S-43.7.1, S-43.7.5, S-1.1.1, S-10.5.1]

SC-2.X.1.2: They must not be misinterpreted as a command to direct an additional controller to provide a control action that is already assigned. [S-37.1.2, S-39.3.2, S-40.4.2, S-41.5.5, S-42.6.2, S-42.6.5, S-1.1.2]

SC-2.X.1.3: They must not be misinterpreted as a command to assign a specific controller to provide a control action it cannot provide, nor as a command to not direct a specific controller that can provide a control action. [S-38.2.2, S-41.5.2, S-41.5.5, S-44.8.2, S-44.8.5]

SC-2.X.1.4: They must not be misinterpreted as a command to disregard hazardous gaps or overlaps in directing controllers to hand off a control action. [S-47.1.6, S-48.2.6, S-56.3.8]

SC-2.X.1.5: They must be directly and unambiguously traceable to the control authority of the team. Specifically:

SC-2.X.1.5.1: They must not be misinterpreted as direction for the team to provide, or not provide, certain control actions when the command is intended to apply to other control actions. [S-1.1.1, S-1.1.2, S-10.5.2]

SC-2.X.1.6: They must not mischaracterize the priority of control actions. [S-10.5.1, S-10.5.3, S-11.6.1, S-11.6.5]

SC-2.X.1.7: They must not be misinterpreted as directions to execute coupled control actions separately. [S-3.3.1, S-3.3.3, S-3.3.5, S-2.2.1, S-2.2.3, S-2.2.5, S-11.6.3, S-4.4.5]

SC-2.X.1.8: They must not be misinterpreted as a direction to provide a control actions coupled to another, even if the other coupled action is not properly executed. [S-3.3.2, S-2.2.2]

SC-2.X.1.9: They must not be misinterpreted as setting temporal constraints that conflict with the safe sequencing of control actions. [S-15.1.6, S-15.1.8, S-17.2.6, S-17.2.8]

SC-2.X.1.9.1: They must not constrain the end time of one action to be before the start time of another coupled action that requires temporal overlap.

SC-2.X.2: ***Unsafe Control Input:*** Controllers outside the team must not issue commands to controllers on the team that conflict with commands provided by the team leadership unless the command is necessary to prevent the system from entering a hazardous state. Specifically:

SC-2.X.2.1: Controllers outside the team must not direct controllers on the team to provide control actions that are different from those directed by the team leadership. Similarly, they must not override controllers on the team from issuing control actions directed by team leadership [S-37.1.3, S-40.4.3, S-43.7.3, S-1.1.3, S-3.3.4, S-2.2.3, S-4.4.3]

SC-2.X.2.2: Controllers outside the team must not direct controllers on the team to issue control actions that have already been assigned to other controllers by the team leadership. [S-38.2.3, S-39.3.3, S-40.4.3, S-41.5.3, S-41.5.4, S-42.6.4] [DA]

SC-2.X.2.3: Controllers outside the team must not direct controllers on the team to issue control actions that they are not capable of performing. [S-44.8.3, S-44.8.4]

SC-2.X.2.4: Controllers outside the team must not direct controllers on the team in a way that conflicts with the safe temporal sequencing of their task executions. Specifically:

SC-2.X.2.4.1: Controllers outside the team must not issue commands that conflict with the safe sequencing of a control handoff between controllers. [S-47.1.7, S-48.2.7, S-56.3.7]

SC-2.X.2.4.2: Controllers outside the team must not issue commands that conflict with the safe sequencing of control actions provided by other controllers. [S-15.1.7, S-17.2.7]

SC-2.X.2.5: Controllers outside the team must not direct controllers on the team in a way that alters the control priorities specified by the team leadership. [S-10.5.4]

SC-2.X.2.6: Controllers outside the team must not introduce tasks into the system (e.g., fake tasks, duplicative tasks) that obfuscate, distract, or deny the ability of controllers on the team to execute tasks directed by the team leadership. [S-10.5.4]